НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6. 050201 «Системна інженерія»

на тему: «Мобільний застосунок взаємодії викладач-студент»

Виконала:
студентка IV курсу, групи ІА-51
Федорчук Любов Борисівна _____

Керівник:
Професор, д.т.н. Ролік О.І.
_____

Рецензент:
Доцент, к.т.н. Попенко В.Д.
_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2019 рік

Пояснювальна записка
до дипломного проекту
на тему: «Мобільний застосунок взаємодії викладач-студент»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ У РОБОТІ СКОРОЧЕНЬ

FTP — File Transfer Protocol — протокол передачі файлів;

HTTP — HyperText Transfer Protocol — протокол передачі гіпертексту;

HTTPS — HyperText Transfer Protocol Secure — безпечний протокол передачі гіпертексту;

JSON — JavaScript Object Notation — нотація об'єкта JavaScript;

MVC — Model-View-Controller — підхід до проектування архітектури Модель-Представлення-Контролер;

MVP — Model-View-Presenter — підхід до проектування архітектури Модель-Представлення-Ведучий;

MVVM — Model-View-ViewModel — підхід до проектування архітектури Модель-Представлення-Модель виду;

SDK — Software Development Kit — набір із засобів розробки;

SQL — Structured Query Language — мова структурованих запитів;

UI — User Interface — інтерфейс користувача;

VIPER — View-Interactor-Presenter-Entity-Router — підхід до проектування архітектури Представлення-Взаємодія-Ведучий-Сутність-Провідник;

БД — база даних;

ОС — операційна система;

ПЗ — програмне забезпечення;

ПК — персональний комп’ютер;

ПП — програмний продукт;

СУБД — система управління базою даних.

ВСТУП

У сучасному світі у зв'язку з ростом потреб людини в обміні інформацією, відповідно, збільшується число методів та засобів комунікації. Проте, з появою нових методів комунікації проблеми перенасичення ринку сервісами, що необхідні для використання в повсякденному житті, набувають особливого значення.

Враховуючи потреби людини в мобільності — смартфон став невід'ємною частиною життя сучасної людини. В результаті чого, з кожним роком число мобільних застосунків, що в середньому в день використовує людина, постійно збільшується. Програми задовольняють потреби користувачів з різними темпами, від щоденних до щомісячних. Зважаючи на велику різноманітність застосунків для комунікації існує велика імовірність того, що інформація не досягне свого кінцевого одержувача. Попри прогрес сучасних технологій людина все ще має обмежені ресурси та концентрацію уваги. Саме тому питання виокремлення єдиного та багатофункціонального засобу взаємодії викладача та студента є актуальним.

Метою даного дипломного проекту є розробка мобільного застосунку для взаємодії викладача та студента, який підвищить рівень комунікації в навчальному закладі надаватиме швидкий та зручний доступ до усіх необхідних навчальних матеріалів і обліку успішності студента.

Досягнення мети даного проекту зумовлюється виконанням наступних задач: розробка зручного та гнучкого засобу взаємодії між користувачами, забезпечення централізованого доступу до навчальних матеріалів, обліку успішності (академічного прогресу), швидкого та гарантованого сповіщення користувачів.

Практична цінність даного проекту полягає в тому, що реалізований програмний продукт, призначений для гнучкої взаємодії різних типів користувачів, а саме викладача та студента. Також передбачена перспектива розширення та розвитку розроблюваного мобільного застосунку, шляхом

впровадження допоміжного функціоналу. Зважаючи на це, отримані результати можна використовувати при розробленні нових версій мобільного застосунку.

Структура даного дипломного проекту включає в себе:

– Аналіз предметної області та аудиторії користувачів, для яких створений даний продукт. В даному розділі детально розглядається предметна область, визначається, яку проблему вирішує мобільний застосунок, складено список вимог, необхідних для визначення масштабу розроблюваного продукту, виокремлена цільова аудиторія та наведено опис типових користувачів застосунку із вказаними характеристиками;

– Аналіз існуючих рішень, що мають повний або частковий функціонал застосунку взаємодії викладача та студента. Для кожного з розглянутих засобів взаємодії були вказані переваги та недоліки для проведення паралелі з вже існуючими та працездатними рішеннями;

– Аналіз підходів та технологій, що були використані в розробці дипломного проекту. Проаналізовано сучасні та прогресивні підходи та технології для розробки мобільних застосунків Для кожного варіанту підходів та технологій було проведено порівняльний аналіз та виявлено найкращі, з розглянутих, рішення відповідно до поставленої задачі;

– Розробка мобільного застосунку. В даному розділі проаналізована архітектура сервісу в цілому, проведено порівняльний аналіз архітектурних підходів до розробки мобільного застосунку відповідно до поставлених вимог, детально описаний розроблений мобільний застосунок та обґрунтовано обрані технічні методи реалізації, вказані всі особливості реалізації окремих компонентів;

– В розділі інструкція користувача наведено розроблений графічний інтерфейс мобільного застосунку, покроково описана інструкція з встановлення мобільного застосунку на пристрій користувача, наведено вимоги до пристрою користувача для коректної роботи із застосунком, описано користувацький інтерфейс та особливості роботи з ним;

– Перспективи подальшого розвитку мобільного застосунку. В даному розділі пропонуються можливі варіанти розширення, наводяться чіткі пропозиції щодо нарощування функціональності мобільного застосунку та запропоновано можливі варіанти полегшення процесу встановлення мобільного застосунку на пристрій користувача;

– Останній розділ дипломного проекту це висновки. У висновках наведено переваги та недоліки розробленого застосунку, зазначено відповідність проекту до поставленої мети.

– У додатках наведено розроблені прототипи мобільного застосунку для двох типів користувачів та додано лістинг розробленої програми із зазначенням відповідних компонентів до яких вони належать.

Графічна частина включає 6 креслеників формату А3. Загальний обсяг 92 сторінки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Попит на мобільні застосунки збільшується з кожним роком та відповідно до прогнозів планується, що до 2020 року мобільні застосунки будуть приносити близько 189 мільярдів доларів США доходів на рік [1]. За статистикою у 2017 році споживачі завантажили 178,1 мільярда мобільних застосунків на власні мобільні пристрої. За прогнозами, в 2022 році цей показник зросте до 258 мільярдів завантажень [2]. На рисунку 1.1 приведена статистика кількості завантажень мобільних програм по всьому світу в 2017, 2018 та 2022 роках.

Рисунок 1.1 — Статистика кількості завантажень мобільних програм

Починаючи з середини 2014 року все більше користувачів почали надавати перевагу смартфону ніж звичному ПК, а в 2016 році користувачів смартфонів стало більше ніж користувачів ПК. Це призвело до гострої

необхідності надавати послуги за допомогою портативних пристроїв, таких як мобільні телефони та планшети. За статистикою, що зображена на рисунку 1.2, в 2017 році як молодь так і старша вікова категорія людей надавали перевагу використанню мобільних телефонів для перевірки електронної пошти [3].

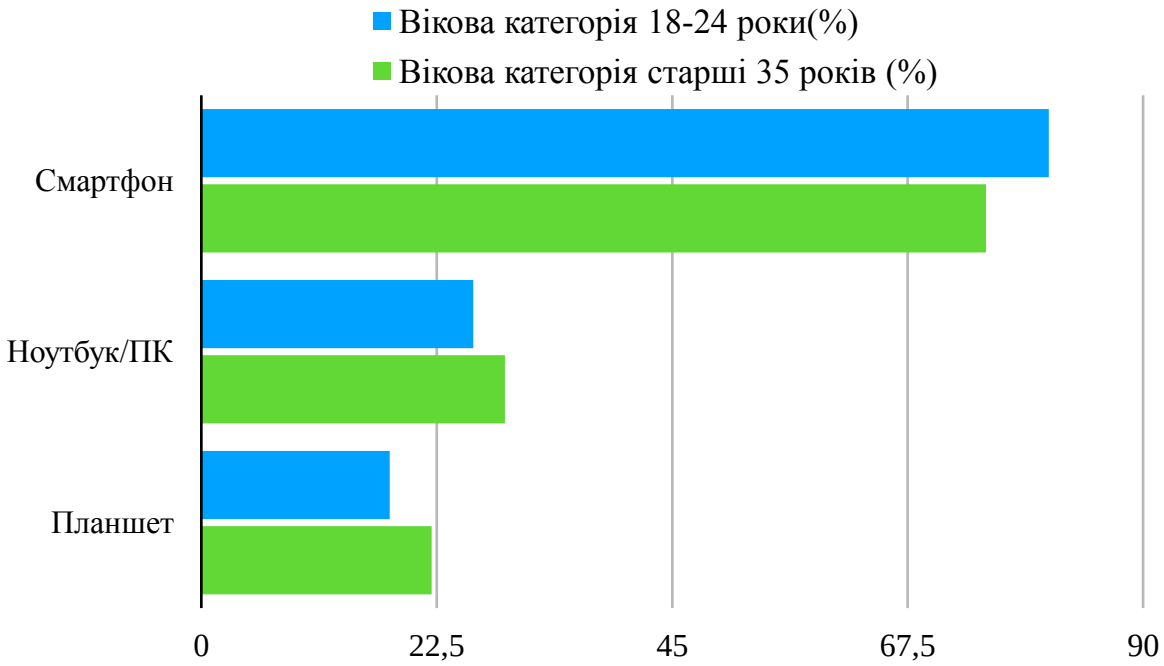


Рисунок 1.2 — Статистика перевірки електронної пошти

Починаючи з 2015 року, росте попит на застосунки для доступу до електронної пошти, програм для підвищення власної продуктивності, наприклад, Google Docs, Slack та Microsoft Office 365. Основною віковою категорією користувачів, що використовують подібні застосунки, є молодь, для котрих мобільний телефон став основним чи навіть єдиним пристроєм для доступу до мережі.

В 2019 році користувачі проводять в мережі 90% власного часу використовуючи саме мобільні застосунки, а лише 10% — користуючись мобільними версіями сайту [4]. Окрім поданої вище статистики, мобільний застосунок має деякі переваги перед сайтом. Однією з основних переваг мобільного застосунку перед сайтом є можливість працювати без

підключення до мережі інтернет, підтримуючи при цьому більшість функціоналу, так само якби пристрій був би підключений до мережі. Мобільний застосунок дає можливість користувачам швидко отримувати доступ до власного контенту, надаючи безперебійну роботу за рахунок збереження значущих даних безпосередньо на мобільному пристрої, до яких також можливо отримати доступ в автономному режимі.

Мобільний застосунок також задовольняє потребу користувача в своєчасному та оперативному наданні значущої інформації. Це є можливим лише за рахунок, реалізованих в операційних системах(ОС) мобільних пристроїв, сповіщень. Однак, сайти не мають такого функціоналу і дають можливість знаходитися завжди на зв’язку.

Якісно реалізований мобільний застосунок може працювати в 1.5 разів швидше за сайт. Така висока швидкодія досягається за рахунок зберігання значимих даних на пристрої, що не вимагає використання великого обсягу пам'яті, але значно прискорює роботу мобільного пристрою.

Варто зазначити, що мобільні застосунки надають індивідуальний підхід відповідно до вподобань користувача, шляхом дотримання стандартів дизайну застосунку. Це значно спрощує та прискорює роботу з застосунком, так як користувач використовує звичний йому інтерфейс.

1.1 Аналіз типів мобільних застосунків

На даний момент, можна виділити три основні типи мобільних застосунків — нативні, веб та гібридні застосунки. Кожний з цих типів має свої переваги та недоліки.

Мобільний веб-застосунок — це оптимізований, під мобільні пристрої, сайт. Він має такі самі переваги та недоліки як і звичний сайт, тому використання цього типу мобільного застосунку в даному дипломному проекті є недоцільним.

Нативний застосунок — це програмне забезпечення (ПЗ) або програмний продукт(ПП), який розроблений для виконання певної задачі в конкретному середовищі або платформі. Головною відмінністю нативних мобільних застосунків від гібридних чи мобільних веб-застосунків є те, що вони розроблені окремо для конкретних пристроїв. В залежності від ОС, для якої вони розроблюються, мобільні застосунки можуть бути написані на мовах програмування, специфічних для цих платформ. Зазвичай використовують Objective-C або Swift для iOS та Java або Kotlin для Android. Перевагами вибору нативних застосунків є те, що таке рішення є швидшим та стабільнішим, коли задача стосується користувацького інтерфейсу. Нативні програми частіше за все мають вищу швидкодію при промальовуванні графічних елементів та анімації, ніж гібридні. Перевагою є те, що нативний застосунок може взаємодіяти через ОС з усіма елементами пристрою, такими як мікрофон, камера, список контактів тощо.

Гібридний мобільний застосунок — це застосунок, що поєднує в собі функціонал як нативних так і мобільних веб-застосунків. З боку функціоналу мобільних веб-застосунків — це вбудований екземпляр веб-браузеру, що використовується для завантаження веб-застосунків прямо з середини власного застосунку.

А з боку нативних рішень — це використання всіх їх переваг, таких як доступ до елементів пристрою через його ОС, розробка з урахуванням звичного інтерфейсу і всіх правил його розробки під конкретну ОС, робота в автономному режимі, за винятком випадку використання сторонніх сайтів.

Використання гібридних мобільних застосунків доцільно лише в тому випадку, коли заздалегідь передбачено використання сторонніх сервісів за допомогою мобільного застосунку. Отже, для досягнення мети даного дипломного проекту доцільна розробка гібридного мобільного застосунку, за рахунок вищезазначених переваг та перспектив потенційного розвитку застосунку.

1.2 Постановка задачі

Даний мобільний застосунок вирішує проблему взаємодії викладача та студента за межами аудиторних занять. Цільову аудиторію мобільного застосунку для взаємодії можна умовно розділити на дві групи — студент і викладач.

Типовим представником групи «студент» є користувач молодшої вікової категорії, від 17 до 27 років, що веде насичений та розсіяний спосіб життя, має низький рівень доходу. У студента переважають такі цінності як суспільне визнання, активне діяльне життя, цікава робота. Має хобі і відводить їм достатньо часу. Для них мобільні технології — невід'ємна частина життя. Ця група розвивається, досягає професійних успіхів і її вплив на економіку й розвиток мобільних технологій теж збільшується. Часто використовують мобільні застосунки для різних цілей, але вважають за краще заходити в них короткими сеансами. Більш охоче використовують месенджери ніж електронну пошту.

Типовим представником групи «викладач» є користувач середньої вікової категорії, від 28 до 44 років, що веде розмірений, спланований, спокійний спосіб життя, має дохід вище середнього. У викладача переважають такі цінності як престижність, соціальна значимість, відповідальність перед студентами, самовдосконалення й самоосвіта. Віддають перевагу мобільним застосункам з дотриманням балансу між ефективністю та комфортом. Проводять достатньо часу в мережі використовуючи, мобільні застосунки, охоче користуються як меседжерами так і електронною поштою.

Задачею дипломного проекту є розробка мобільного застосунку для взаємодії викладача та студента. Розроблений мобільний застосунок має відповідати таким заданим вимогам:

– Надавати доступ до перегляду розкладу занять без необхідності в підключенні до мережі інтернет;

– Забезпечити одноразову авторизацію користувача зі збереженням особистих даних у шифрованому вигляді безпосередньо на власному пристрої;

– Забезпечити управління календарем та надавати можливість створення подій безпосередньо в застосунку;

– Надавати доступ до файлового сховища для збереження, читання та поширення навчальних матеріалів з можливістю використання матеріалів в автономному режимі роботи;

– Забезпечувати синхронізацію локальних даних при відсутності підключення до мережі;

– Підтримувати функціонал обміну миттєвими повідомленнями в режимі реального часу;

– Надавати можливість управління особистим обліковим записом;

– Надавати допоміжний функціонал для використання застосунку, такий як перегляд новин навчального закладу;

– Інтерфейс користувача має бути інтуїтивно зрозумілий та зручний у використанні з дотриманням стандартів та вимог щодо дизайну відповідно до обраної мобільної ОС.

Для досягнення мети даного дипломного проекту для розробки було обрано мобільну ОС iOS та мову програмування Swift.

2 ОГЛЯД ІСНУЮЧИХ ЗАСОБІВ ВЗАЄМОДІЇ ВИКЛАДАЧА ТА СТУДЕНТА

На сьогоднішній день взаємодія викладача та студента в ході навчального процесу виходить далеко за межі аудиторних занять, а саме включає в себе обмін навчальними матеріалами, надання доступу до методичних забезпечень, ведення поточного контролю та успішності студента, розкладу аудиторних занять та надання інформації про проведення запланованих заходів.

Для задоволення кожного з наведених вище запитів, використовуються різні застосунки, які тільки частково покривають задані потреби викладача і студента.

Одними з найбільш відомих засобів взаємодії викладача та студента є Електронний Кампус КПІ, різноманітні служби обміну миттєвими повідомленнями, такі як Telegram, Viber, Skype, WhatsApp, хмарні сховища з можливістю спільного доступу до матеріалів, а саме Dropbox, GoogleDrive, OneDrive та окремі сервіси для перегляду розкладу, новин та подій в навчальному закладі. Безліч відомих зарубіжних університетів також мають власні мобільні додатки, такі як Texas A&M University, Notre Dame Mobile, CarolinaGO. Нижче наведений огляд найбільш відомих засобів взаємодії, кожний в своїй сфері, й описані їх переваги та недоліки.

2.1 Особливості системи «Електронний кампус КПІ»

Інформаційно-телекомунікаційна система «Електронний кампус» — це прикладне програмне забезпечення, яке є елементом інформаційно-телекомунікаційного середовища університету та використовується для інформаційної підтримки повсякденної діяльності студентів, викладачів, співробітників університету, а так само для інформаційної підтримки всіх видів інноваційної діяльності в університеті [5]. Електронний кампус

призначений для інформатизації навчального процесу університету з метою підвищення якості навчання студентів та для забезпечення навчального процесу сучасними інформаційними технологіями. Вікно «Методичне забезпечення» Електронного кампусу у браузері та панель керування зображені на рисунку 2.1.

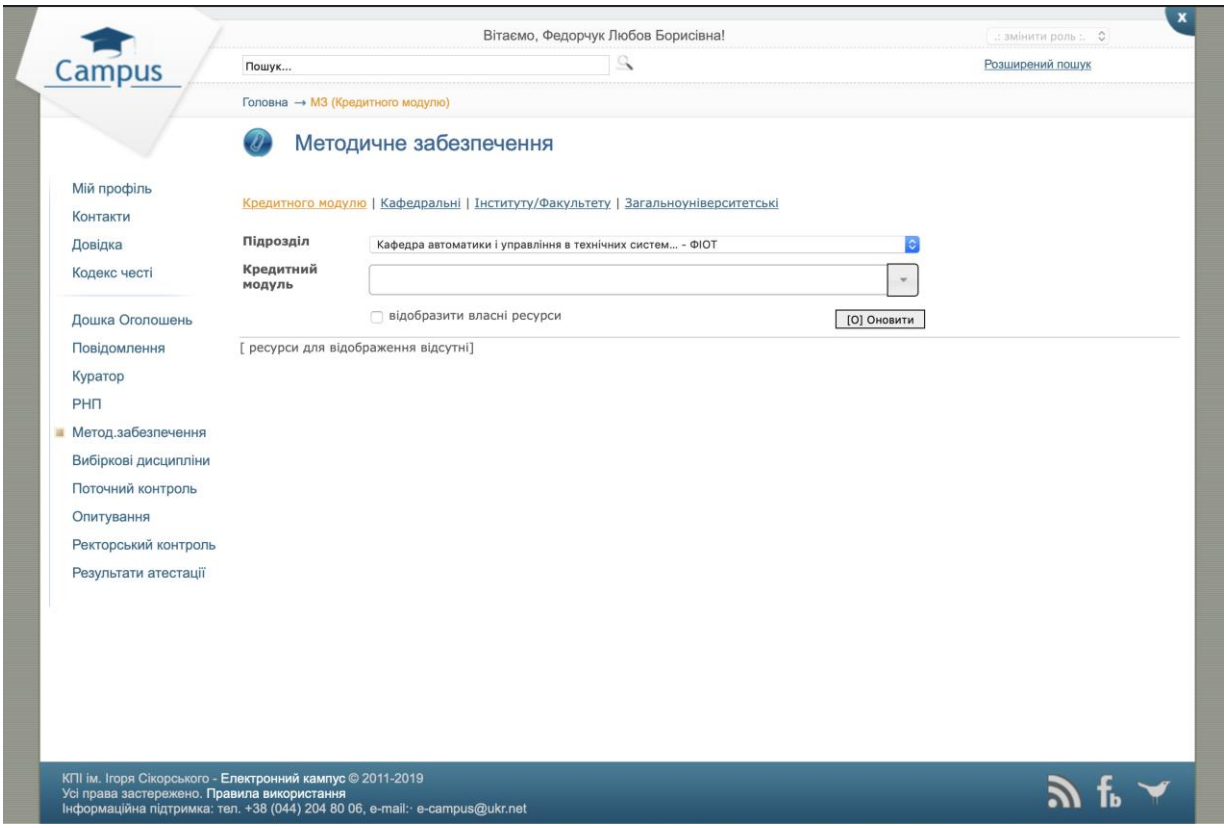


Рисунок 2.1 — Вікно «Методичне забезпечення»

Основні функції:

– Багатобічна комунікація між студентами, викладачами, науковим співтовариством;

– Формування єдиного інформаційного ресурсу, що відображає стан науково-освітнього процесу університету;

– Розповсюдження інформації про майбутні події і заходи та інші довідкові відомості;

– Забезпечення централізованого і зручного доступу до відомостей про діяльність ректорату і підрозділів університету;

– Забезпечення навігації по всьому інформаційному наповненню Електронного кампусу.

Переваги системи «Електронний кампус КПІ»:

– Відсутність необхідності у завантаженні та встановленні додаткових програм для користування цим сервісом. Це значно економить час користувача та ресурси мобільного пристрою, так як немає необхідності в пошуку сервісу для скачування та установки на особистий пристрій. Також відсутня необхідність в налаштуванні сервісу під конкретний пристрій;

– Наявність доступу до поточного контролю студента. Призначений для ведення та контролю успішності студентів, як самими студентами так і викладачами та кураторами груп. Це дозволяє проводити контроль протягом семестру, за підсумками сесії, надає прозорий та об’єктивний доступ балів студента, результатів іспитів, створений для запобігання помилок при ведені поточного контролю та підвищує безпеку документообігу використанням індивідуальних паролів користувачів;

– Наявність доступу до методичних забезпечень та навчальних матеріалів. Надається централізований доступ до навчальних ресурсів та можливість розміщення власних інформаційних ресурсів, таких як статті, методичні вказівки до лабораторних робіт, конспекти лекцій тощо, а також завантажувати матеріали інших користувачів, відповідно до повноважень;

– Наявність інструкції користувача. Наявність інструкції дозволяє користувачеві, який вперше маєте справу з даним сервісом, розібратися в його роботі. В інструкції розкрита повна функціональність та принцип роботи сервісу, після ознайомлення з якою користувач не буде мати проблем під час роботи з сервісом.

Основні недоліки системи «Електронний кампус КПІ» наведені нижче.

– Складний та застарій інтерфейс, з яким потенційний користувач не може розібратися для вільного орієнтування під час використання даного сервісу;

– Відсутність повної сумісності з усіма популярними браузерами, такими як: Google Chrome, Safari, Internet Explorer та Edge, Mozilla Firefox, Opera;

– Відсутність мобільного застосунку, а також мобільної чи адаптивної версії сайту, що ускладнює навігацію та процес використання даного сервісу. Це робить неможливим отримання своєчасного та оперативного доступу до потрібних користувачу даних з мобільного пристрою;

– Необхідність в постійному доступі до мережі для роботи сервісу та відсутність офлайн-режиму. Здавалося б в сучасному світі це вже не вважається суттєвим недоліком та попри все ще досі є необхідність підтримки офлайн-режиму в разі виникнення непередбачуваних ситуацій з боку користувача, коли доступ до мережі відсутній;

– Встановлення обмеження на максимальний розмір для завантаження одного файлу — 2 Мб. В реаліях таке обмеження є недоцільним, бо в середньому, розмір файлу, що містить навчальні матеріали — 4 Мб і більше;

– Відсутність функціоналу миттєвого обміну повідомленням. Спосіб обміну повідомленнями між користувачами влаштований на подобу ведення переписки за допомогою електронної пошти. Таким чином, такий спосіб обміну повідомленнями повторює недоліки електронної пошти, а саме негарантований час пересилки повідомлень, можливість доступу для третіх осіб під час пересилання, відсутність зворотного зв’язку у разі прочитання повідомлення користувачем чи втратою листа.

Даний сервіс взаємодії викладача та студента має як безліч очевидних недоліків, так і деякі переваги. З усіх описаних варіантів Електронний кампус є найкращим рішенням на даний момент, який задовольняє більшість потреб

користувачів. Однак, наведені вище недоліки сервісу значно ускладнюють роботу з ним, так як деякі недоліки являються критичними для користувачів.

2.2 Особливості мобільного застосунку Texas A&M University

Texas A&M University — це мобільний застосунок, який надає різноманітну інформацію про університет [6]. Застосунок має каталог, який допомагає студентам легко знаходити контакти в кампусі. Він також має модулі курсу, які дозволяють студентам здійснювати пошук і планувати розклад занять. Головна сторінка мобільного застосунку Texas A&M University для ОС iOS зображена на рисунок 2.2.

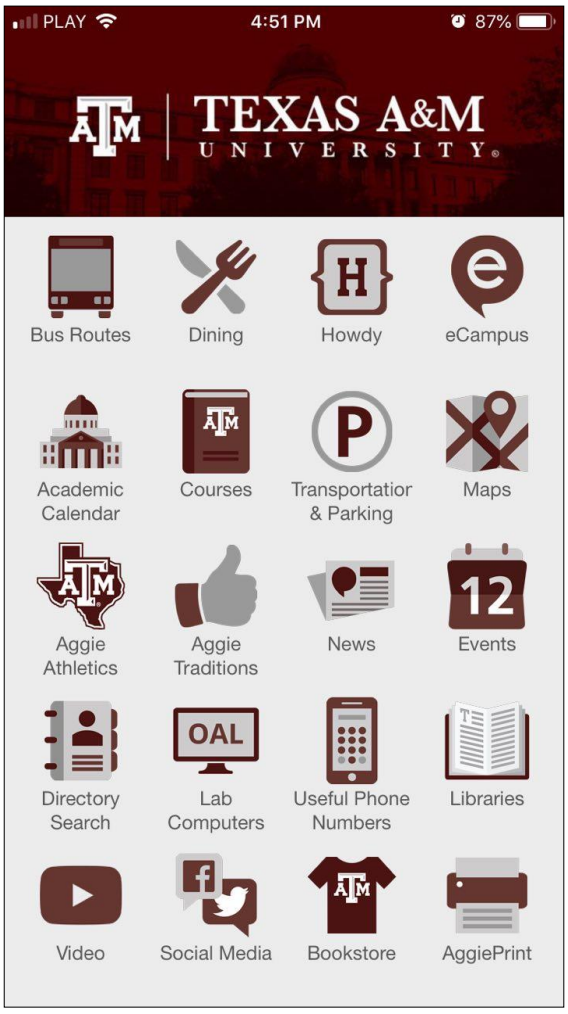


Рисунок 2.2 — Головна сторінка застосунку Texas A&M University

Переваги:

– Наявність інтерактивної карти кампусу, що значно полегшує пересування користувача по студентському містечку та між корпусами, спрощує пошук потрібної аудиторії, паркінгу для автомобілів та закладів харчування, що розташовані на території університету;

– Наявність централізованого та швидкого доступу до навчальних матеріалів, які можна переглядати в застосунку;

– Наявність розкладу автобусів та автобусних маршрутів, що прямують до університету та від університету до різних точок в місті;

– Можливість перегляду останніх нових університету;

– Наявність актуальної інформації про майбутні події і заходи, що плануються в університеті, можливість додавати майбутні заходи до календарю та створювати нагадування про них.

Недоліки:

– Відсутність розкладу аудиторних занять;

– Відсутня можливість завантаження та розміщення інформаційних ресурсів, таких як статті, конспекти лекцій тощо;

– Відсутність автономного режиму, без підключення до мережі;

– Відсутність функціоналу обміну повідомленням. Даний застосунок не передбачає методів ведення листування, хоч і має можливість зберігати контактні дані користувачів, а саме номери телефонів, адреси електронної пошти тощо та переглядати список номерів телефону, що можуть бути корисними;

– Не передбачено використання веб-інтерфейсу, на разі існує підтримка тільки мобільних ОС iOS та Android;

– Відсутній доступ до поточного контролю студента.

Наведений мобільний застосунок є дуже потужним інструментом, що надає достатній функціонал для виконання найбільш часто виконуваних операцій. Проте, найбільшим недоліком такого рішення є те, що даний

застосунок розроблений тільки для Техаського університету A&M та не може бути використаний іншими навчальними закладами.

2.3 Особливості сервісу хостингу файлів Dropbox

Dropbox — це файловий хостинг, що дозволяє користувачеві розміщувати та зберігати файли за допомогою клієнта або через браузеру, підтримує безпечну синхронізацію файлів на всіх пристроях, дає можливість надавати іншим користувачам доступ до обраних файлів [7]. Домашня сторінка мобільного застосунку Dropbox для ОС iOS зображена на рисунок 2.3.

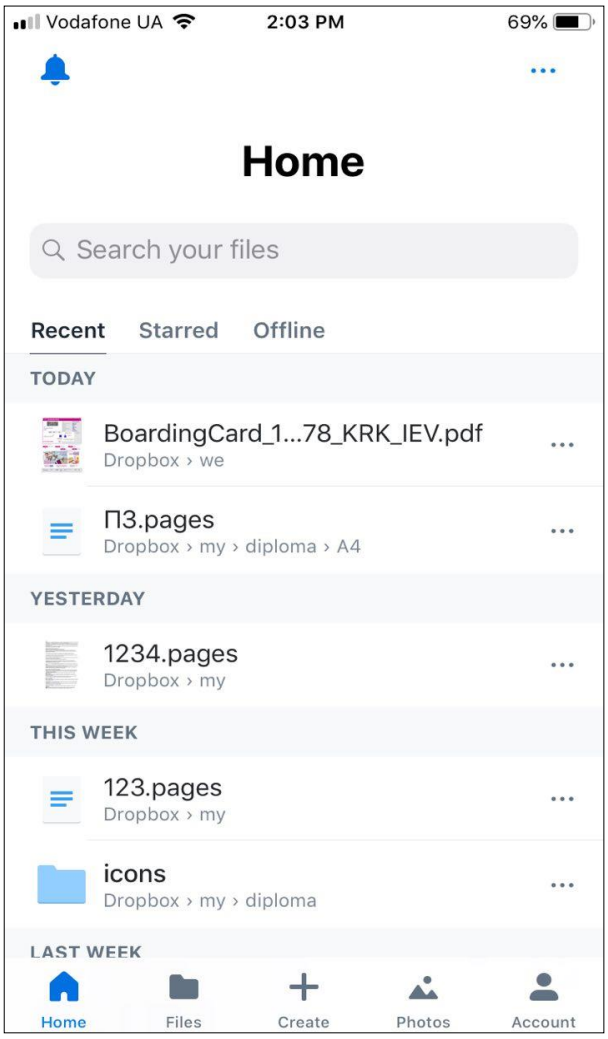


Рисунок 2.3 — Домашня сторінка мобільного застосунку Dropbox

Нижче наведені основні переваги.

– Надання швидкого та централізованого доступу до інформаційних ресурсів;

– Наявність доступу через веб-інтерфейс з підтримкою повного функціоналу;

– Підтримка ОС Microsoft Windows, macOS, Linux;

– Наявність мобільного додатку, підтримка мобільних ОС iOS, Android та Windows Phone, BlackBerry;

– Підтримка синхронізації файлів на всіх пристроях;

– Доступне розмежування прав доступу до папок, спільна робота з файлами;

– Наявність офлайн-режиму для обраних файлів. Користувач має можливість завантажити обрані файли для подальшого використання без підключення до мережі;

– Присутня функція сповіщення та перегляду чужих змін у файлах спільного доступу;

– Відсутнє обмеження для одного завантаження через клієнтський застосунок;

– Ведення історії завантажень. Це дає можливість користувачу відновити дані після видалення файлів з сервера;

– Dropbox зберігає всі видалення та історію версій файлів користувача протягом 30 днів не займаючи виділений обсяг пам’яті для збереження даних;

– Наявна можливість створення, завантаження та редагування документів Word, Excel, PowerPoint та текстових файлів безпосередньо за допомогою веб-інтерфейсу та мобільного застосунку;

– Dropbox підтримує перегляд файлів у більшості форматах.

Недоліки файлового сховища наведено нижче:

– Максимальний обсяг збережених даних — 2 ГБ безкоштовно з можливістю розширення до 1ТБ;

– Наявне обмеження для одного завантаження через веб-інтерфейс в 20 ГБ;

– Встановлено обмеження на розмір файлів для попереднього перегляду в залежності від його формату;

– Низька швидкість завантаження файлів великого обсягу.

Загалом, Dropbox є найкращим рішенням для зберігання, обміну та надання навчальних матеріалів. Проте, як було зазначено вище, надання матеріалів та методичних забезпечень є тільки частиною, яка входить в широке поняття взаємодії викладача та студента в сучасному світі. Тому дане застосунку не може повністю задовольнити запити викладача та студента, але може слугувати прикладом того, який саме функціонал є необхідним для створення якісного продукту.

2.4 Особливості сервісу обміну миттєвими повідомленнями Telegram

Telegram — сервіс обміну миттєвими повідомленнями або месенджер, що дозволяє обмінюватися повідомленнями й мультимедійними файлами багатьох форматів [8]. Відмінність таких повідомлень від електронної пошти полягає в тому, що обмін повідомленнями відбувається в реальному часі. Обмін повідомленнями доступний як між двома користувачами так для групи користувачів.

Також існує підтримка каналів — це «чати», що дозволяють відправляти повідомлення необмеженому числу користувачів, які можуть лише переглядати отримані повідомлення. Основними перевагами сервісу обміну повідомленнями Telegram є:

– Наявність мобільного додатку, підтримка мобільних ОС iOS, Android та Windows Phone;

– Підтримка ОС Microsoft Windows, macOS, Linux;

– Наявність доступу через веб-інтерфейс з підтримкою повного функціоналу;

– Підтримка синхронізації на всіх пристроях;

– Telegram не має обмежень щодо кількості особистих та групових чатів, каналів;

– Відсутнє обмеження на максимальний обсяг збережених даних;

– Є можливість здійснювати дзвінки;

– Наявність зручного та інтуїтивно зрозумілого для користувача інтерфейсу;

– Наявність миттєвих сповіщень користувача про повідомлення або дзвінки за умови наявності підключення до мережі;

– Наявність статусу в користувача (в мережі, був в мережі деякий час назад);

– Можливість відключати сповіщення для обраних особистих чи групових чатів та каналів;

– Відсутнє обмеження на кількість повідомлень в чатах;

– Підтримка обміну голосовими повідомленнями.

З основних недоліків можна виділити такі:

– Наявне обмеження на максимальний розмір файлу, що може відправити користувач за один раз — 1.5 Гб;

– Максимальна кількість учасників у групі — 5000 користувачів;

– Встановлено обмеження на максимальну довжину одного повідомлення — 4096 символів;

– Наявне обмеження на тривалість голосового повідомлення — 60 хвилин.

Описаний вище месенджер є одним з найкращих сервісів, який доступний на ринку на сьогоднішній день. На рисунку 2.4 показана домашня сторінка мобільного застосунку Telegram для ОС iOS. Наявні функції

повністю задовольняють вимоги користувачів, що користуються сервісами обміну миттєвими повідомленнями.

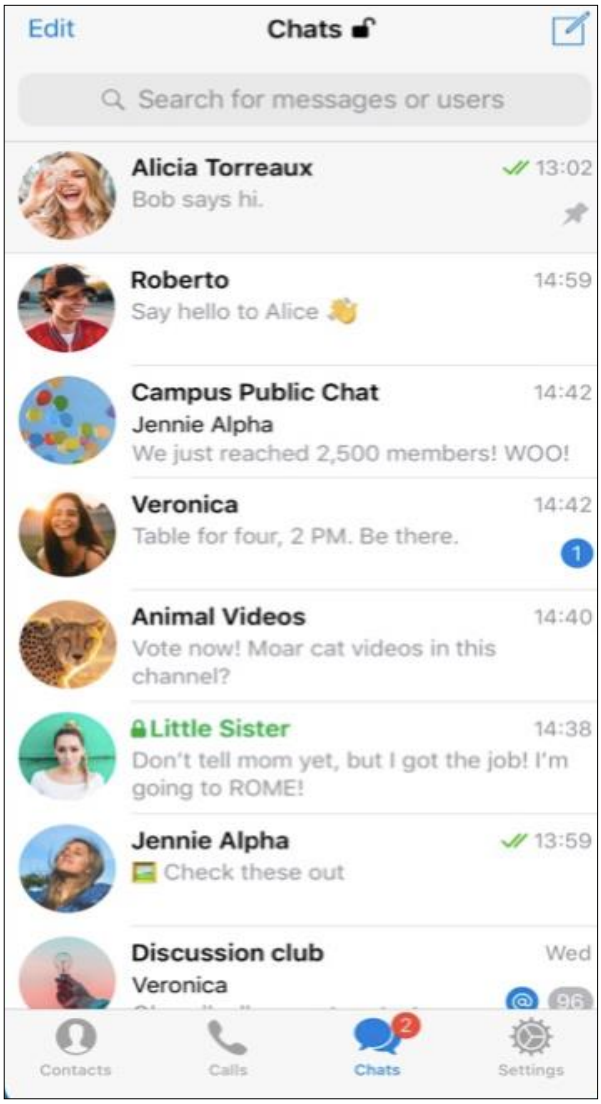


Рисунок 2.4 — Домашня сторінка мобільного застосунку Telegram

Але так само як і з файловий хостингом даний сервіс не може повністю задовольнити потреби взаємодії викладача та студента. Telegram може слугувати прикладом та основою для створення мобільного застосунку для взаємодії викладача та студента, враховуючи те, що він мене суттєві переваги та незначні недоліки.

3 ВИБІР ПІДХОДІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ

3.1 Огляд існуючих мобільних операційних систем

На сьогодні, можна виокремити дві найбільш поширені мобільні ОС, на які припадає понад 99% ринку, такі як iOS та Android.

iOS — це мобільна ОС, що належить компанії Apple. Спочатку мала назву iPhone operating system, але пізніше була змінена з появою нових пристроїв iPad. Дана ОС має широкий спектр вбудованих функцій, що повністю задовольняють потреби сучасного користувача. Мобільні застосунки, що розробляється під дану ОС стають доступними користувачу тільки пройшовши декілька етапів тестування та перевірки на несправності. Це гарантує високу якість застосунку, відсутність критичних несправностей та відповідність користувацького інтерфейсу всім стандартам та вимогам, що описані компанію Apple та викладені в документі, що має назву Human Interface Guidelines. Саме тому всі застосунки, що створюються під дану платформу інтуїтивно зрозумілі та зручні у використанні.

Остання стабільна версія даної ОС, яка була випущена — 12.0, проте після цього ще було випущено три оновлення до версії 12.3. Користувачі будь-якого пристрою, випущеного компанією Apple, завжди отримують останні оновлення та мають змогу їх встановити на свої пристрої, а усі виправлення критичних несправностей стають доступними користувачам найшвидше. Хоча iOS є закритою системою, проте дана ОС вважається більш захищеною, ніж альтернативні системи. За рахунок своєї закритості та захищеності стороннім особам практично неможливо отримати доступ до персональних даних користувачів.

Компанія Apple повідомила, що з 2021 року стане можливе використання одного застосунку, що створений для ОС iOS, на всіх пристроях — iPhone, iPad та Mac.

Android — це мобільна ОС, що належить компанії Google та розроблена базуючись на ядрі Linux. На відміну від iOS, Android є відкритою

системою. На момент 2019 року загальна кількість застосунків, що були випущені під платформу Android налічує 2.1 мільйонів, а під платформу iOS — 1.8 мільйонів [9]. Проте в світі існує тенденція, що під ОС iOS мобільні застосунки розробляються в першу чергу. Для мобільних застосунків під ОС Android немає таких жорстких вимог, як під ОС iOS, лише рекомендації до дизайну ПЗ, що створені та викладені в загальний доступ компанією Google, та мають назву — Material Design. Це дозволяє розробникам ПЗ створювати унікальні застосунки з винятковим дизайном, але це також в рази збільшує час, який користувач повинен буде витратити для того, щоб розібратись з абсолютно новим інтерфейсом та принципом роботи.

Для того, щоб мобільний застосунок дійшов до кінцевого користувача немає необхідності проходити жодних етапів тестування та перевірок, достатньо лише завантажити застосунок у Google Play. Після цього будь-який користувач зможе завантажити програму на власний мобільний пристрій. З одного боку, це значно спрощує процес надання користувачу програмного продукту, з іншого боку, це створює ряд проблем. Таке спрощення разом з відкритістю системи не захищає користувача від зловмисників, дає можливість отримати доступ до персональних даних та елементів системи.

Остання стабільна версія ОС, що була випущена компанією Google — 9.0 Pie. Для мобільних пристроїв на базі ОС Android існує поширена проблема, а саме можливість оновлення версії ОС тільки на одну версію вище. Подальше оновлення часто є неможливим. Саме тому великий відсоток мобільних пристроїв ще й досі працюють на старих версіях ОС, які вже не підтримуються. Новий функціонал для таких пристроїв є недоступним, користувачі перестають отримувати оновлення мобільних програм, значна частина мобільних застосунків перестають працювати після того, як версія ОС знята з підтримки. Відповідно до статистики, що подана на офіційному сайті для розробників ПЗ під ОС Android лише на 10.4% мобільних пристроїв встановлено останнє оновлення версії ОС, а 42.1%

мобільних пристроїв ще й досі встановлені 6.0 і нижче версії ОС, які більше не підтримуються [10].

Рішення кросплатформної розробки також не є прийнятним рішенням, коли в основі розробки мобільного застосунку перш за все стоїть висока планка якості, а саме безпека користувацьких даних та персоналізації застосунку. Це можливо досягти лише за умови розробки мобільного застосунку під конкретну ОС окремо.

Зважаючи на описані вище переваги та недоліки розглянутих ОС було прийняте рішення розробки мобільного застосунку для взаємодії викладача та студента під мобільну ОС iOS.

3.2 Огляд засобів розробки мобільних застосунків

В ході огляду предметної області в розділі 1 була обрана розробка гібридного мобільного застосунку та спираючись на всі вищезгадані переваги була обрана ОС iOS. На сьогоднішній день, розробка нативних або гібридних мобільних застосунків під ОС iOS можлива на декількох мовах програмування — Swift та Objective-C.

Objective-C — це заснована на класах об'єктно-орієнтована мова програмування, реалізована як розширення мови C. Мову програмування Objective-C було обрано в якості основної мови, використовуваної NeXT для ОС NeXTSTEP, з якої отримано macOS та iOS [11]. На момент випуску, Objective-C включала в себе безліч функції, що лише починали впроваджуватися у інші мови програмування або просто були відсутні. Objective-C вважалась практичною мовою, в тому розумінні, що використовувалась тонка середа виконання програми, написана на C, яка незначно збільшувала розмір програми. Objective-C сумісна з C++ і Objective C++. Це дозволяє використовувати ці мови разом, що в свою чергу, збільшить можливості розробника та розширить функціонал, що може бути реалізований в застосунку. Objective-C вважається стабільною мовою, так як

останнє оновлення до версії 2.0 було випущено у 2006 році. Це свідчить про те, що остання версія має усталений функціонал та повну відсутність помилок. Проте дана мова є доволі старою, має складний синтаксис, що породжує помилки при написанні ПП, обмежений функціонал та нижчу швидкодію, порівняно з сучасними мовами програмування чи їх оновленими версіями.

До 2014 року Objective-C була основною мовою програмування для розробки ПП під macOS, проте того ж року, компанія Apple випустила нову мову програмування — Swift.

Swift — це мова програмування загального призначення, створена з використанням сучасного підходу до безпеки, продуктивності й шаблонам проектування ПЗ [12]. На сьогодні, вже випущена версія Swift 5.0. Синтаксис мови доволі простий, що полегшує читання та написання коду програм. Для реалізації одного і того ж функціоналу, використовуючи Swift, кількість рядків написаного коду буде значно меншою порівняно з Objective-C. Це досягається за рахунок того, що розробники мови Swift відмовилися від застарілих стилістичних принципів, наприклад, крапка з комою, в кінці рядка або ж круглі дужки, які оточують вирази в межах операторів умови — if else.

Одним з основних пріоритетів розробників мови Swift була спрямованість на безпеку. Конструкція Swift відкидає декілька типів помилок, що все ще можливі в Objective-C. Саме тому, застосунки, що розроблені на цій мові стабільніші та менш схильні до помилок й збоїв.

На відміну від Objective-C, Swift — це мова з відкритим вихідним кодом. Вихідний код викладений у спільний доступ на веб-сервісі для хостингу IT-проектів GitHub, де кожен бажаючий може його переглянути та внести власний вклад в розробку. Це також дозволяє компанії Apple отримувати відгуки від спільноти розробників, що використовують Swift, та вносити поліпшення на основі цих відгуків.

Крім того, швидкодія Swift майже така ж, як і в C ++, котра вважається найшвидшою в області арифметичних обчислення алгоритмів. Swift працює

швидше, ніж Objective-C, тому що при розробці були зняті обмеження мови C та було проведено поліпшення за допомогою передових технологій, які були ще недоступні при розробці мови C.

В 2018 році компанія JetBrains проводила опитування серед розробників, котрі обрали Swift або Objective-C в якості однієї з трьох основних мов програмування. За статистикою, що зображена на рисунку 3.1, більше половини розробників використовують лише Swift в якості основної мови програмування [13].

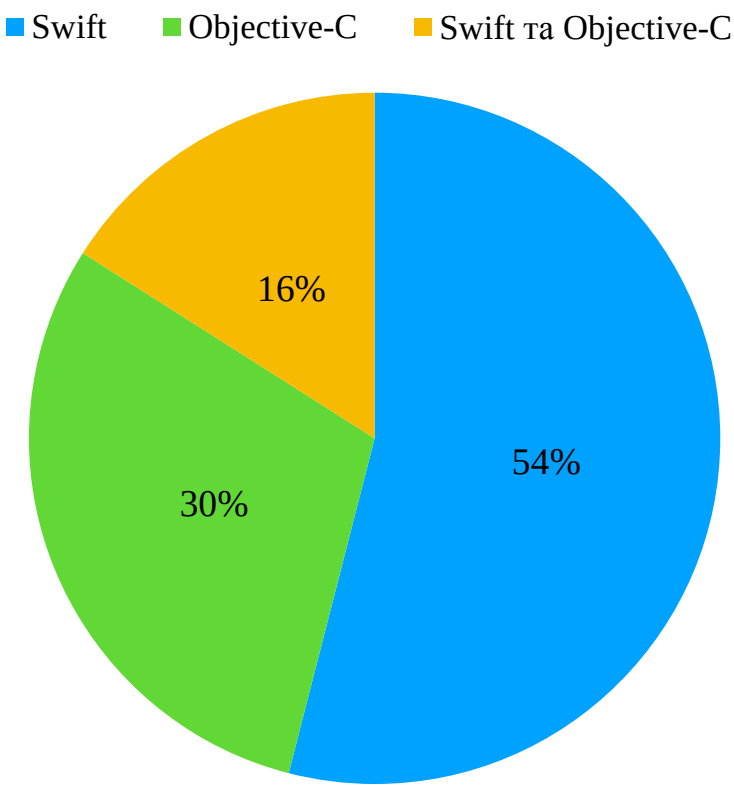


Рисунок 3.1 — Статистика по мовам програмування

З усіх розробників, що використовують лише Objective-C або обидві мови одразу, 29% розробників планують найближчим часом перейти на Swift.

Саме описані переваги та поширеність Swift обумовили вибір даної мови програмування для створення мобільного застосунку взаємодії викладач-студент.

В якості середи розробки було використано інтегроване середовище Xcode версії 10.2.1, яке випущено компанією Apple та доступне повністю безкоштовно, на відмінну від середовища AppCode від компанії JetBrains. Xcode 10 включає в себе все необхідне для створення застосунків для всіх платформ Apple.

Редактор вихідного коду дозволяє легше перетворювати або реорганізовувати код, бачити зміни в управлінні вихідним кодом й швидко отримувати детальну інформацію про відмінності у вихідному коді [14]. Xcode включає в себе механізм тестування, вбудований в нього. Можливий запуск модульних тестів, а також тестів користувацького інтерфейсу та продуктивності одночасно на декількох пристроях.

Для безперервного налаштування інтеграції можливий запуск декількох різних типів імітованих пристроїв, щоб виконати весь тестовий комплект від початку до кінця. У програмний пакет входять емулятори пристроїв випущених компанією Apple, які підтримуються, наприклад, різні моделі iPad, iPhone, Apple Watch тощо.

4 РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

4.1 Особливості клієнт-серверної архітектури сервісу

В основі розробки та роботи сучасного ПЗ лежить найпоширеніша клієнт-серверна архітектура. Ця концепція має на увазі під собою розділення функціоналу, обов'язків з обробки даних та відповідальності між двома компонентами — клієнтами та серверами, що взаємодіють між собою. На рисунку 4.1 показано схематичне зображення архітектури клієнт-сервер.

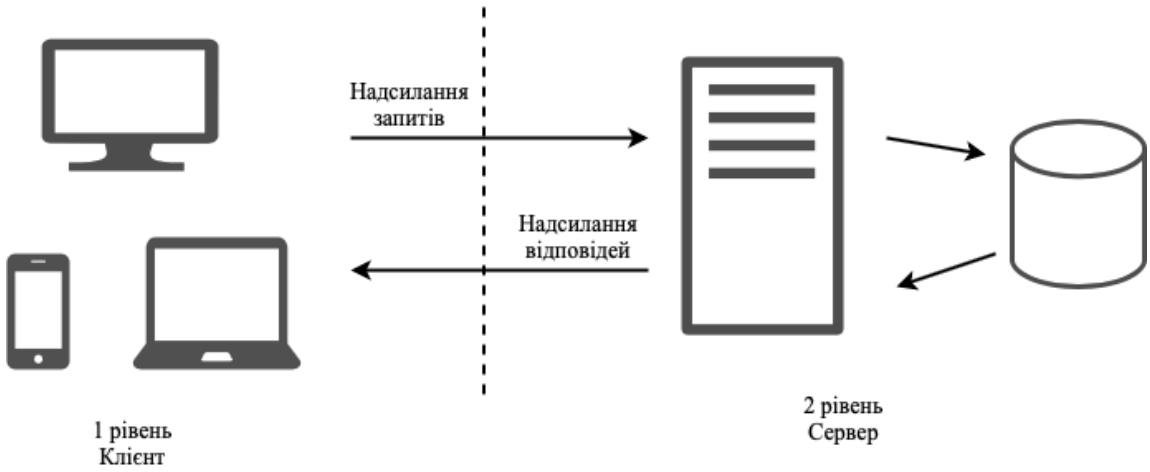


Рисунок 4.1 — Схема архітектури клієнт-сервер

Нижче наведено опис області відповідальності та функцій цих компонентів.

Клієнт, або перший рівень — це рівень представлення, що виконує функцію представлення даних та отримання запитів від користувача через клієнтський інтерфейс. Якщо розглядати клієнта як мобільний застосунок, тоді на клієнті лежить відповідальність за реалізацію функціоналу основної роботи мобільного застосунку та обробки даних на ньому. Також кожен клієнт, як правило, може мати власне локальне сховище, для збереження даних та надання доступу до них. Клієнти, в свою чергу, поділяються на тонкі та товсті клієнти.

Тонкий клієнт — це клієнтське ПЗ, що спроектовано так, щоб основна частина обробки даних відбувалася на сервері. Зазвичай, такі клієнти мають менший розмір та вищу швидкодію. Але під цим не мається на увазі загальна простота застосунку, в тонких клієнтах часто роблять акцент на користувацькому інтерфейсі та його функціональних можливостях. Класичним прикладом тонкого клієнта є веб-браузер. З переходом вбік надання послуг з використанням хмарних платформ також призвів до переходу до тонких клієнтів.

Товстий клієнт — це клієнтське ПЗ, що на противагу тонкому клієнту, має більш широку функціональність незалежно від сервера. Основними відмінностями є те, що функціональність застосунку не залежить від підключення до мережі, тобто реалізована безперебійна робота застосунку без підключення або з повільним підключенням. В таких випадках сервер часто використовується як сховище для збереження та синхронізації даних, однак товстий клієнт, зазвичай, має власне локальне сховище. Товстий клієнт має ряд недоліків, таких як: великий розмір клієнтського застосунку, потребують більше ресурсів від клієнтських пристроїв, зростають проблеми безпеки та складність реалізації ПЗ. Отже, розглядаючи клієнт як мобільний застосунок, вигідніше розробляти його як тонкий клієнт за рахунок обмеженості ресурсів мобільних пристроїв. Таким чином, тема роботи розуміє під собою розробку клієнтського мобільного застосунку для клієнт-серверного застосунку.

Сервер, або другий рівень — це компонент архітектури клієнт-сервер, що виконує сервісні функції по запитам клієнтів, обробляє та зберігає дані, надає доступ до певних ресурсів і послуг. Сервер, зазвичай, містить основну бізнес-логіку застосунку, реалізує взаємодію зі сховищем даних, відповідає за безпеку всього застосунку. Серверна частина застосунку може бути розроблена за допомогою певних архітектурних підходів, наприклад, мікросервісна або монолітна архітектура.

Мікросервісний архітектурний підхід — це архітектурний шаблон, що припускає відмову від єдиної, монолітної структури застосунку. Схематичне зображення структури мікросервісної архітектури показано на рисунку 4.2. Основні задачі серверної частини розподіляються між декількома невеликими, наскільки це можливо, слабо пов’язаними, високо узгодженими та легко змінюваними модулями — мікросервісами, які взаємодіють між собою по мережі. Такий підхід робить акцент на легку масштабованість, за рахунок чіткого розділення задач та слабкої зв’язаності сервісів, простоту їх розгортання, високу швидкодію та стійкість до збоїв. Проте такий підхід підвищує складність розробки та підтримки застосунків.

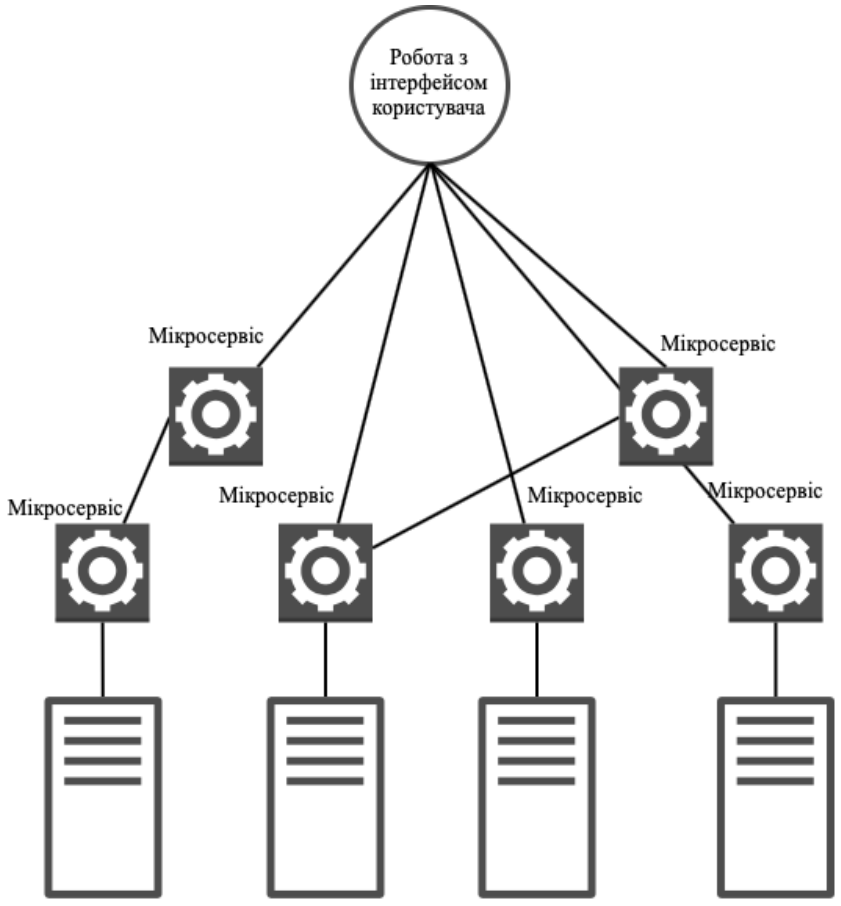


Рисунок 4.2 — Структура мікросервісної архітектури застосунку

На противагу мікросервісному підходу, монолітний архітектурний підхід — це єдиний, комплексний модуль, поділений на окремі, але взаємопов’язані рівні. Схематичне зображення структури монолітної

архітектури показано на рисунку 4.3. Серверна частина з використанням такого підходу умовно та функціонально поділена на різні сервіси, але тільки в межах єдиного застосунку.

Монолітний архітектурний підхід значно прискорює швидкість розробки застосунку, має високу зв’язаність, що підвищує швидкодію застосунку та забезпечує вищу безпеку. Монолітна архітектура набагато простіша в реалізації, управлінні та розгортанні. Однак, використовуючи даний підхід з розширенням функціоналу застосунку буде зростати складність підтримки і розширення серверної частини, буде з’являтися більше помилок та збоїв, за рахунок складності тестування сервісів.

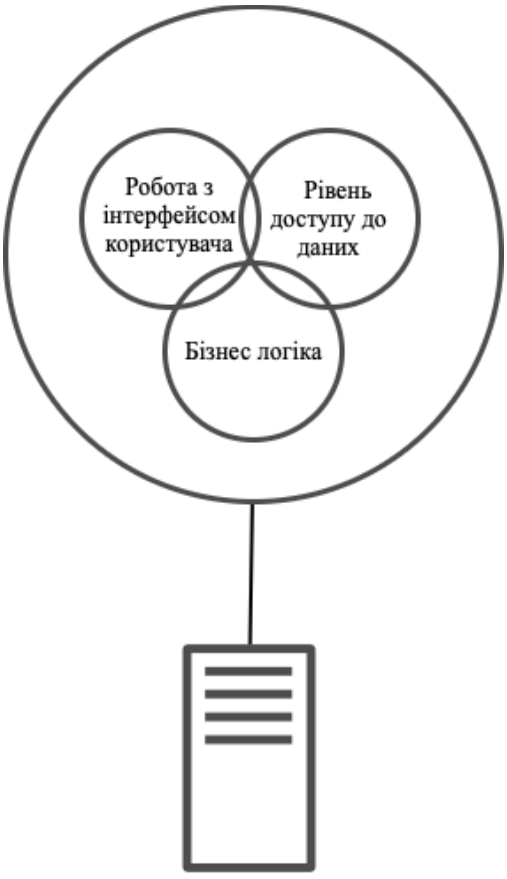


Рисунок 4.3 — Структура монолітної архітектури застосунку

Взаємодія клієнта і сервера відбувається по мережі за допомогою різних мережевих протоколів, наприклад, HTTP, HTTPS чи FTP, шини даних чи систем обміну повідомленнями.

Найпоширеніший метод взаємодії — це взаємодія між компонентами клієнт-серверної архітектури за допомогою протоколу HTTP, що відбувається шляхом пересилання HTTP повідомлень — HTTP запитів та HTTP відповідей. Такі повідомлення містять в собі один із методів, який визначає необхідну дію. Основні методи наведені нижче:

– GET — метод, що виконує запит на отримання даних від сервера до якого іде запит;

– POST — метод, основне завдання якого, передача клієнтських даних серверу;

– PUT — метод, що використовується для передачі оновлених клієнтських даних серверу, якщо вони вже існують на сервері, в протилежному випадку створює їх;

– DELETE — метод для видалення даних з сервера.

Повідомлення також має вказаний шлях, зазвичай посилання, що визначає цільовий сервер чи ресурс та додаткову інформацію, яка може передаватися у зашифрованому вигляді. Клієнт, для отримання чи надсилання даних відправляє HTTP запит, який містить всю необхідну, для сервера, інформацію, на який сервер надсилає свою HTTP відповідь. Така відповідь містить результат запиту відповідно до використаного методу, а також спеціальний код стану, який дозволяє клієнту дізнаватися про результат запиту до сервера. Найпоширеніші коди стану наведені нижче.

– Код 200 OK вказує на успішність запиту клієнта;

– Код 301 (Переміщено назавжди, Moved Permanently) означає, що дані, на які клієнт надсилав запит, знаходяться по іншому посиланню на постійній основі;

– Код 404 (Не знайдено, Not Found) вказує на те, що запит розпізнаний успішно, проте дані по вказаному посиланню були не знайдені;

– Код 500 (Внутрішня помилка сервера, Internal Server Error) означає будь-яку внутрішню помилку сервера, що не може бути описана іншими кодами станів.

Протокол HTTPS є розширенням протоколу HTTP та існує для підтримки шифрування даних, що передаються від сервера до клієнта, з метою безпеки. Головною відмінністю є те, що всі основні елементи піддаються шифруванню, а саме шлях до ресурсу, параметри запиту та додаткова інформація, що часто містить дані користувача. Такий підхід дозволяє створити шифрований канал зв'язку, реалізувати авторизацію користувачів, тобто обмежити доступ до серверу для неавторизованих сторонніх користувачів. Проте, це також не гарантує абсолютної захищеності даних на сервері, лише дає можливість безпечного обміну даними, так як вилучити дані з серверу простіше ніж перехопити їх при передачі.

Ще одним методом є взаємодія клієнта й сервера за допомогою менеджера черг. Менеджер черг або брокер повідомлень — це ПЗ для роботи з чергами повідомлень, основна задача якого приймати та передавати повідомлення. Проводячи аналогію з протоколом HTTP, повідомлення можуть містити будь-яку інформацію. Менеджер черг може приймати повідомлення від клієнта та зберігати до тих пір, поки сервер не підключиться до нього. Після цього менеджер черг вилучить повідомлення з черги та передасть його серверу на відповідну обробку. Менеджер черг може використовуватися не тільки для взаємодії клієнтів та серверів, але й для розподілення навантаження між сервісами, за умови використання мікросервісної архітектури. Найпоширенішими є такі менеджери черг:

– ActiveMQ — є найбільш популярним і мультипротокольним сервісом обміну повідомленнями на основі Java з відкритим вихідним кодом. Він підтримує стандартні галузеві протоколи, тому користувачі отримують переваги вибору клієнтів на широкому діапазоні мов та платформ [15]. Наявна підтримка протоколів OpenWire (підтримка мов Java, C, C++, C#), Stomp (підтримка мов Ruby, Perl, Python, PHP), AMQP, MQTT;

– RabbitMQ — є легким менеджером черг, простим в розгортанні на хмарних платформах [16]. Підтримує кілька протоколів обміну

повідомленнями — STOMP, AMQP, MQTT, HTTP. RabbitMQ має велику кількість готових розширень з можливістю створення власних розширень.

При взаємодії клієнта й сервера відбувається обмін даними, які можуть бути представлені в певному форматі. Найбільш розповсюджений формат даних для обміну та зберігання — JSON.

JSON (JavaScript Object Notation) — текстовий формат для обміну повідомленнями, простий в читанні і розумінні людиною, повністю незалежний від мови програмування, проте може використовуватися майже всіма мовами програмування. Більшість мов програмування по замовчуванню містять в собі механізм для роботи з даним форматом. Зазвичай, при розробці клієнт-серверної архітектури застосунку, призначені для користувача дані зберігаються на стороні сервера — найпоширенішим рішенням є збереження в базах даних (БД).

Відносини в БД можна розглядати як «математичне безліч», що містить в собі число атрибутів, які сумарно становлять собою базу даних і інформацію, що зберігається в ній. Робота з БД відбувається за допомогою системи управління базами даних, що розташовується на сервері разом з БД та здійснює доступ до БД безпосередньо, в монопольному режимі.

Системи управління базами даних (СУБД) — спеціальні програми (або бібліотеки) для управління базами даних різних розмірів й форм, забезпечує надійне, безпечне збереження та цілісність даних [17]. Сучасні СУБД відповідають таким критеріям як надійність, тобто забезпечення не тільки високої відмовостійкості, а й можливість відновлення втрачених даних у разі неочікуваного збою, паралелізм, тобто забезпечення паралельного та конкурентного доступу до даних. PostgreSQL, MySQL, SQLite, Oracle Database, Microsoft SQL Server — найпоширеніші СУБД, що використовують при розробці застосунків з клієнт-серверним підходом до розробки архітектури.

4.2 Підходи до розробки архітектури мобільних застосунків

У зв'язку з підвищенням вимог й зростанням складності мобільних застосунків під ОС iOS підвищується і його архітектурна складність. Стандартні архітектурні рішення вже не можуть задовольнити всі вимоги до проектування застосунків. Саме тому, у відповідь на підвищення вимог до розробки застосунків з’являються нові підходи та шаблони для розробки архітектури. Серед них варто розглянути сучасні підходи до проектування архітектури мобільних застосунків — Модель-Представлення-Модель виду(Model-View-ViewModel, MVVM), Представлення-Взаємодія-Ведучий-Сутність-Провідник(View-Interactor-Presenter-Entity-Router, VIPER), Модель-Представлення-Ведучий (Model-View-Presenter, MVP) та класичний MVC.

MVC — це класичний підхід до проектування архітектури мобільного застосунку, що рекомендований до використання компанією Apple. Схема шаблону MVC представлена на рисунку 4.4.

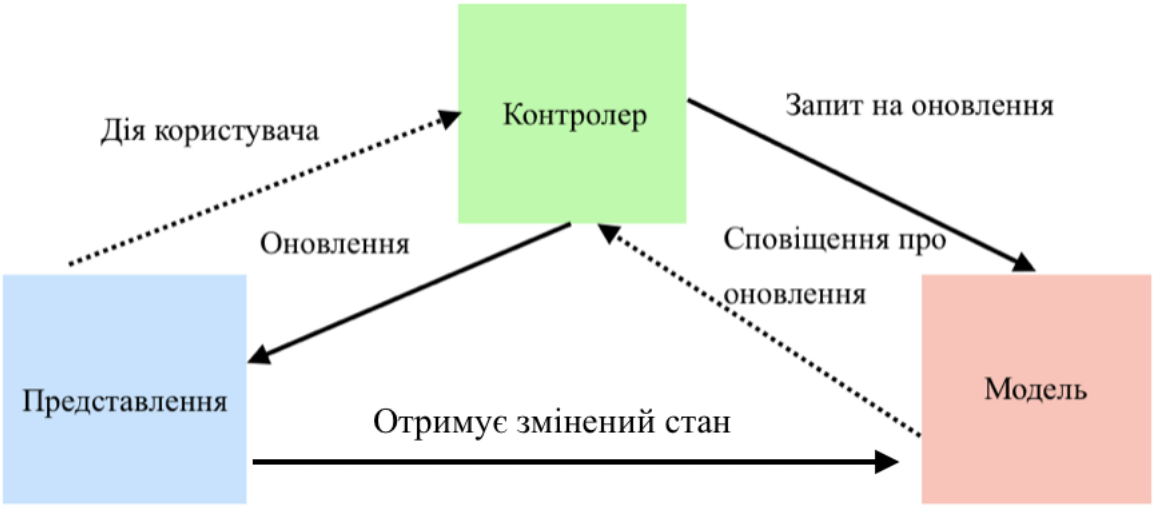


Рисунок 4.4 — Схема підходу MVC до проектування архітектури

Представлення — це візуальна проекція моделі, а саме користувацький інтерфейс, що відображається користувачу в певний момент часу.

Модель — це дані, які будуть відображатися в компонентах представлення, а контролер — це міст, що пов’язує модель та представлення, інакше кажучи, це з'єднання між користувачем та системою. Виходячи з цього, дана архітектура має ряд суттєвих переваг: простота реалізації, а отже швидший процес розробки. Відсутність дублювання коду та висока швидкість роботи застосунку за рахунок підтримки асинхронності. Проте, даний підхід має певні недоліки, а саме — слабке розділення відповідальності між архітектурними одиницями. Весь функціонал перекладається на контролер. В майбутньому, це може породити проблему масштабування та ускладнити модульне тестування мобільного застосунку.

MVVM — це альтернативний підхід до проектування архітектури, який використовує модель виду у якості посередника між моделлю та представленням. Схема шаблону MVVM представлена на рисунку 4.5. Модель та представлення мають таке ж призначення як і у шаблону MVC. Модель виду відповідає за представлення даних з моделі та перетворення форматів даних, проте не впливає на зміну елементів представлення, який відповідає за користувацький інтерфейс. Підхід MVVM схожий на нижчеописаний MVP, але для MVVM зв'язування представлення з моделлю виду відбувається автоматично, а для MVP даний зв'язок доводиться програмувати.

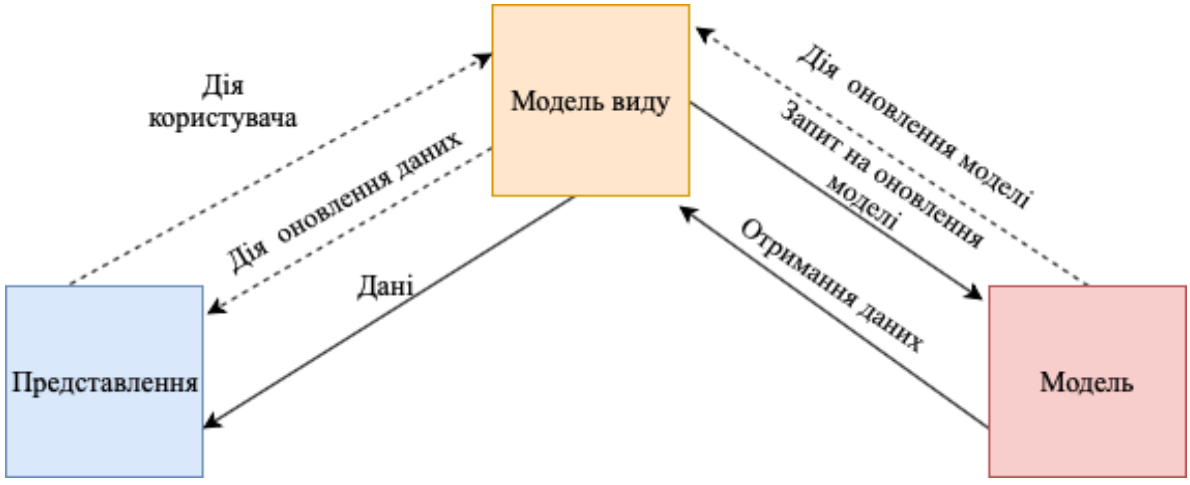


Рисунок 4.5 — Схема MVVM до проектування архітектури

Проте архітектурний шаблон MVVM не позбавлений недоліків, які випливають із його переваг. Для досягнення автоматичного зв'язування представлення та моделі доводиться впроваджувати додаткові фреймворки, наприклад, ReactiveCocoa, що значно збільшує розмір застосунку, час розробки та об'єм роботи.

MVP — це розширення шаблону MVC від Apple, де відповідальність контролера розподіляється між представленням і ведучим. Схематичне зображення основних принципів архітектурного підходу MVP представлено на рисунку 4.6. Використовуючи такий підхід ведучий, на відміну від контролера, стає незалежним від представлення, через це зникає сильна зв'язаність архітектурних одиниць. Це досягається шляхом впровадження шаблону — інверсія контролю (Inversion of Control), який використовується для зменшення зв'язаності окремих компонентів програми. Хоча ведучий все ще розділяє часткову відповідальність з представленням, а саме відповідає за оновлення представлення відповідно до нових даних й станів.

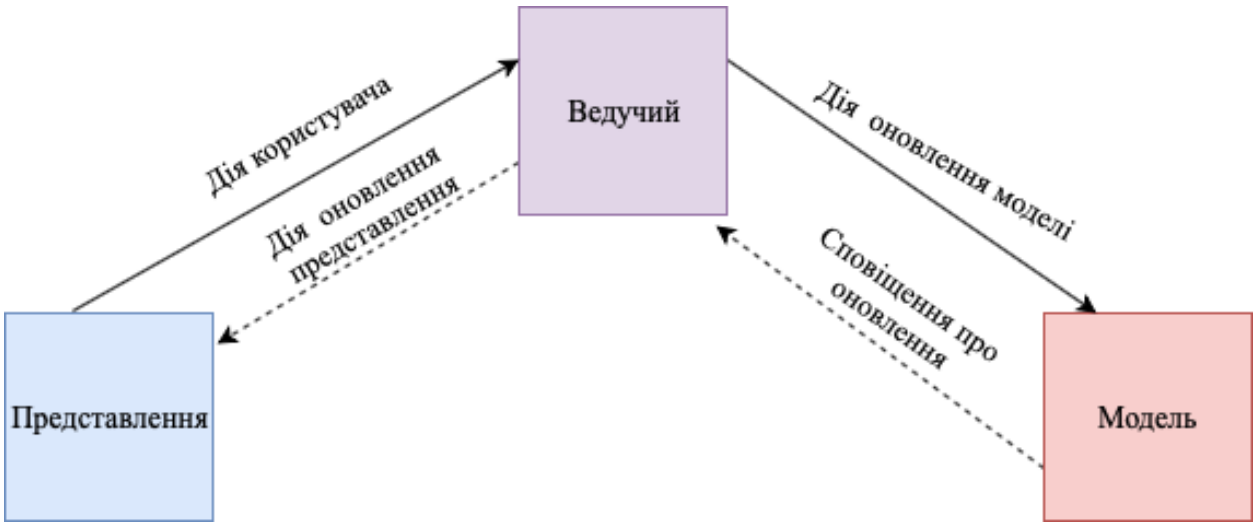


Рисунок 4.6 — Схема MVP до проектування архітектури

Основною причиною для використання даного архітектурного підходу є спрощення модульного тестування застосунку, проте, за рахунок необхідності впровадження шаблону інверсії контролю уповільнюється процес розробки. Через те, що між представленням та ведучим

встановлюється зв’язок один-до -одного, то комплексні та складні представлення можуть мати декілька ведучих, що збільшує об’єм коду і, відповідно, розмір мобільного застосунку.

VIPER — це найновіший підхід до розробки архітектури мобільного застосунку. На відмінну від вищеописаних шаблонів, VIPER поділяється на п’ять архітектурних компонентів. Таке розділення має на меті знизити зв’язаність коду, спростити маршрутизацію в застосунку. Схема шаблону VIPER представлена на рисунку 4.7. Компоненти представлення та ведучий мають такі ж обов’язки як і в MVP. Компонент сутність — це об’єкти даних, які описують структурні одиниці застосунку, а компонент взаємодія відповідає за доступ до даних, тобто до компоненту сутність та схожий за обов’язками з компонентом модель у вищеописаних архітектурних шаблонах. Провідник — це архітектурна одиниця, що відповідальна за переходи та маршрутизацію між всіма компонентами.

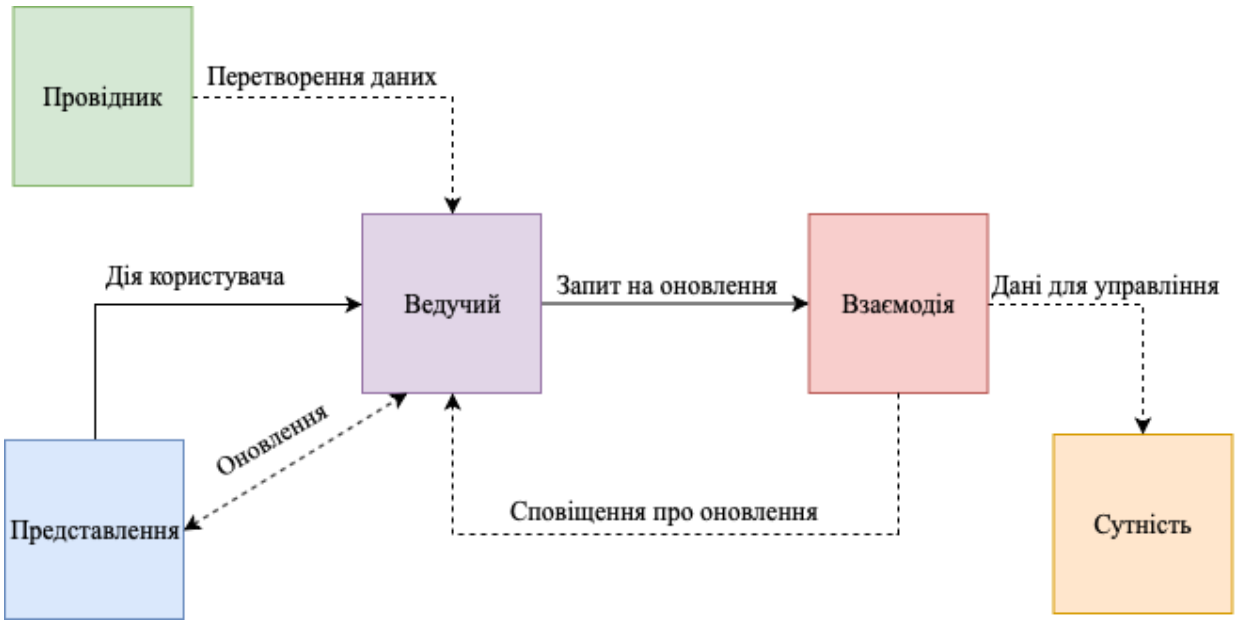


Рисунок 4.7 — Схема підходу VIPER до проектування архітектури

З використанням такого підходу можна досягти повної незалежності архітектурних компонентів, вирішення проблеми навігації в середині застосунку та підвищити рівень тестування. Проте такий підхід значно

збільшує кількість коду, й відповідно, збільшує розмір самого застосунку, підвищує складність розробки, але спрощує підтримку та масштабування застосунку в майбутньому.

Для розробки мобільного застосунку взаємодії викладач-студент було обрано архітектурний шаблон — MVC. Дотримання правил шаблону MVC дозволяє забезпечити модульність, швидший процес розробки, більш легке обслуговування та розширення застосунку. Діаграма компонентів мобільного застосунку взаємодії викладач-студент наведена на кресленику Д6.

4.3 Розробка прототипу мобільного застосунку

Створення прототипу є одним з основних етапів розробки мобільного застосунку. Прототип — це модель, в якій окреслено основний функціонал та концепцію застосунку. За даними, що надаються платформою для аналізу мобільних застосунків Quettra відомо, що в середньому мобільний застосунок втрачає 77% активних користувачів протягом перших 3 днів після встановлення [18]. Протягом 30 днів цей показник збільшується до 90% всіх активних користувачів.

Для уникнення подібної ситуації, відразу після визначення цільової аудиторії й проблем, які вирішує застосунок, що розробляється, починається етап прототипування. Такий похід дозволяє знизити ризики, визначити основну функціональність та загальну концепцію мобільного застосунку, а також дозволяє зберегти час та гроші, які могли бути втрачені при розробці неперевірених на практиці рішень.

Прототипи бувають різних типів, в залежності від специфічних задач, які вони вирішують, хоча основне призначення залишається сталим. Основними вважаються такі типи прототипів:

– Концептуальний прототип, що призначений для схематичного зображення екранів мобільного застосунку, що планується. Зазвичай, це найперший прототип, що створюється для обрису загальної ідея застосунку.

Може бути створений на папері, за допомогою стікерів чи інших підручних матеріалів;

– Інтерактивний прототип, що призначений для опису поведінки, тестування сценаріїв дій користувача мобільного застосунку що буде розроблений;

– Каркасний прототип призначений для відображення екранів мобільного застосунку, а саме графічне зображення, що містить ключові елементи призначеного для користувача інтерфейсу (поля, кнопки, значки тощо). Забезпечує візуальну індикацію застосунку, відображає основні деталі, розмір елементів і їх розташування;

– Анімований прототип — це прототип високого рівня, що створюється для відображення взаємодії застосунку з користувачем, візуалізація якого створюється за допомогою анімації;

– Макетний прототип — це візуальне відображення майбутнього застосунку, в якому відсутні інтерактивні дії, проте такий прототип майже повністю відображає роботу застосунку.

Крім того, часто буває достатньо створити один прототип змішаного типу для досягнення основної мети прототипування. Спочатку був створений концептуальний прототип на папері для структурування всіх ідей, що плануються реалізовуватися в мобільному застосунку. Це дозволило візуалізувати більшу частину запланованого функціоналу, оцінити складність застосунку, обрати відповідну архітектуру та переконатися у правильності обраних методів та засобів для розробки.

В дипломному проекті був розроблений прототип змішаного типу, а саме каркасний та інтерактивний. Для відображення основних сценаріїв дій було обрано два типи користувачів, для яких буде розроблений даний застосунок — викладач та студент. Прототип, який відображає можливі сценарії та загальну картину застосунку з боку викладача наведено в додатку А. Прототип містить головний екран мобільного застосунку взаємодії

викладач-студент з якого починаються всі можливі сценарії дій користувача. На рисунку 4.8 показаний сценарій перегляду розкладу викладачем.

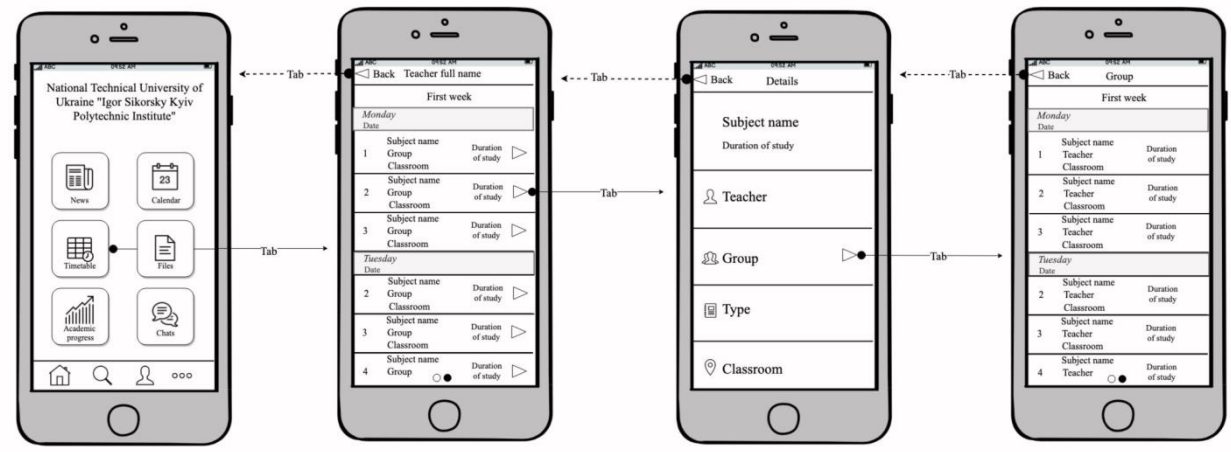


Рисунок 4.8 — Сценарій «перегляд розкладу занять»

З головного екрану користувач може перейти на екран перегляду розкладу занять натиснувши на елемент «Timetable» на екрані, після чого на екрані буде відображатися розклад занять на поточний тиждень. Користувач може дізнатися про деталі обраного заняття натиснувши на нього. Після чого відбудеться перехід на третій екран, на якому будуть відображені додаткові відомості про обране заняття, а саме назва предмету, викладач, група, номер аудиторії та тип заняття — практичне, лекційне чи лабораторне.

Користувач, в поточному сценарії це викладач, може переглянути розклад групи, натиснувши на секцію «Group». Після чого відбудеться перехід до наступного екрану, на якому буде показаний розклад групи. Для повернення на попередній екран користувач може натиснути на кнопку «Back» вгорі екрану після чого відбудеться перехід до попереднього екрану. Така дія буде відбуватися до тих пір, поки користувач не перейде на головний екран.

Для користувача студент сценарій перегляду розкладу занять реалізований аналогічним чином. Ряд сценарії, що показані в прототипах реалізовані однаково для обох типів користувачів, а саме: перегляд новин

навчального закладу, перегляд календаря та деталей про заплановані події, пошук по мобільному застосунку, листування в чатах та налаштування облікового запису користувача. Сценарій, що відображає дію перегляду календаря показаний на рисунку 4.9.

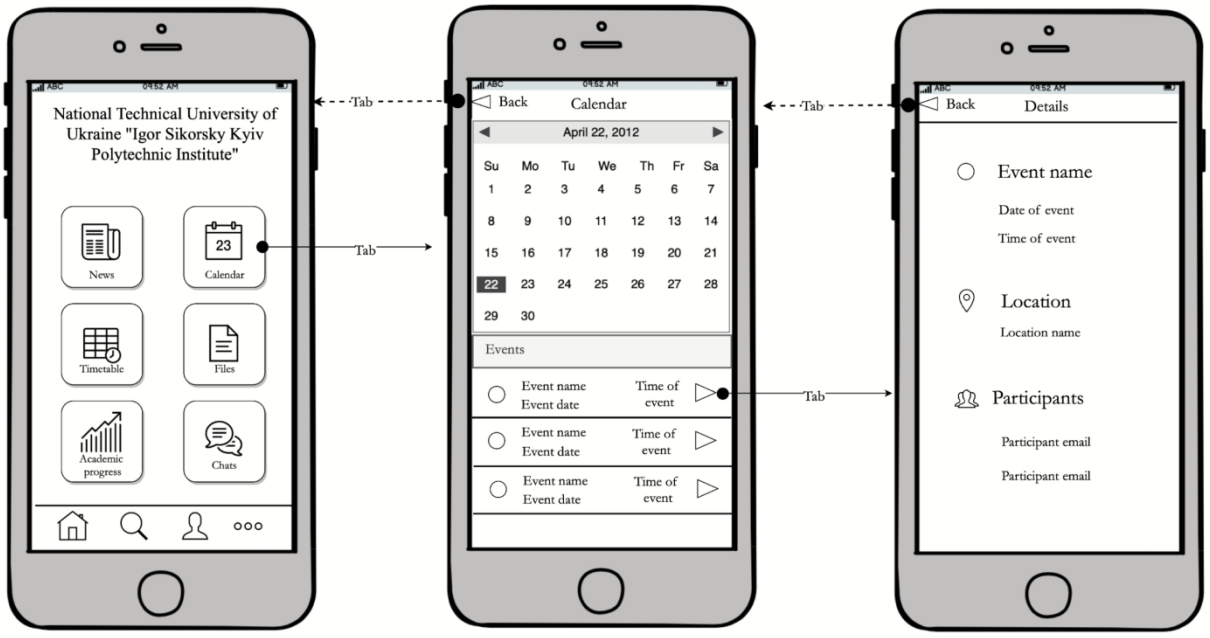


Рисунок 4.9 — Сценарій «перегляд календаря»

Перехід з головного екрану буде відбуватися аналогічним чином, як при перегляді розкладу занять. Прототип, який відображає можливі сценарії та загальну картину застосунку з боку студента наведено в додатку Б. Після оцінки та аналізу створеного прототипу стає можливим перехід до безпосередньої розробки мобільного застосунку взаємодії викладач-студент.

4.4 Розробка застосунку взаємодії викладач-студент

Застосунок, що розроблений в рамках дипломного проекту призначений для забезпечення зручного та гнучкого засобу взаємодії викладача та студента в межах навчального процесу. Як було зазначено у 3 розділі, для розробки застосунку була обрана мобільна ОС iOS та мова

програмування Swift. Обраною архітектурою для мобільного застосунку є MVC. Розроблений застосунок можна умовно поділити на окремі компоненти, що взаємодіють між собою та кожен з яких несе відповідальність за виконання відведених для них задач.

4.4.1 Розробка компоненту авторизації

Розроблений компонент авторизації забезпечує вхід користувача в застосунок з-під власного облікового запису з використанням імені користувача або його електронної пошти та пароля. Ця задача вирішена в класі UserService. В ньому реалізовано функціонал надсилання запиту на сервер за допомогою HTTP методу — GET. Для реалізації необхідних HTTP методів імпортується бібліотека Alamofire. Alamofire — це мережева HTTP-бібліотека на базі Swift для iOS та macOS. Бібліотека надає інтерфейс поверх мережевого стека Apple Foundation, спрощує ряд загальних мережевих задач й забезпечує гнучкість вихідного коду. Об’єкт класу UserService містить функцію, так як мова програмування Swift оперує саме функціями, а не методами, getUser, яка власне реалізує запит на отримання даних користувача. Вихідний код представлено у додатку Б.1.

Для початкового входу в застосунок необхідно мати підключення до мережі, тому під час авторизації користувача здійснюється перевірка підключення мобільного пристрою до мережі. Ця функція викликається при початковому завантаженні представлення екрану з привітанням в класах WelcomeViewController та Connectivity. Якщо ж пристрій не підключений, тоді користувачу відображається сповіщення про це та рекомендація про здійснення підключення. Вихідний код представлений у додатку Б.2. Також при завантаженні представлення відбувається перевірка чи користувач вже авторизований.

Якщо дані користувача були збережені на пристрої, то на цьому процес авторизації завершується та користувач може продовжувати роботу з застосунком.

Функціонал авторизації та взаємодії з користувачем реалізований в класі SignInViewController, вихідний код якого наведено у додатку Б.3. Саме в цьому класі виконується GET запит до серверу, шляхом створення об’єкта класу UserService. Виклик функції авторизації відбувається у відповідь на дію користувача — натискання кнопки авторизації. Після зчитування введених даних користувача створюється HTTP заголовок, що надсилається в запиті. Реалізована обробка ситуації, коли користувач не ввів усі необхідні для авторизації дані або ввів їх неправильно. Користувачу виводиться сповіщення відповідно до помилки.

Заголовки HTTP — це поле даних, що надає потрібну інформацію про HTTP запити, HTTP відповіді або про об'єкт, що відправлений в тілі запиту. Існує чотири типи заголовків HTTP-повідомлень — загальний заголовок, заголовок запиту клієнта, заголовок відповіді сервера, заголовок сутності. В проекті використовуються тільки заголовки клієнта, які містять параметр авторизації. Значення поля заголовка запиту авторизації складається з облікових даних, що містять інформацію про аутентифікацію користувача для області запитуваного ресурсу. Специфікація HTTP / 1.0 визначає схему авторизації як BASIC, де ключем авторизації є пароль та ім’я користувача чи електронна адреса, що закодовані з використанням схеми кодування даних Base64. Приклад заголовку авторизації показано на рисунку 4.10.

KEY	VALUE
Authorization: Basic	ZWMyMTQyZTEtYjY2OC00NjNjLTkwMTMtYzg0NTA0Zj

Рисунок 4.10 — Заголовок авторизації

Якщо запит пройшов успішно та отримана відповідь містить код стану 200ОК й відповідний об’єкт у форматі JSON, тоді користувацькі дані, що

необхідні для авторизації, зберігаються в шифрованому вигляді у Keychain. Приклад об’єкту User у форматі JSON показаний на рисунку 4.11, а вихідний код моделі User наведено у додатку Б.4. Аналогічним чином створені структури для моделей, що відповідальні за шар доступу до даних, а саме Article, Timetable, File, Event, Message, Channel.


```
{
   "success":"true",
   "value":{
      "username":"lbfedor",
      "email":"lb.fedor@gmail.com",
      "phone_number":"+38(099)0723426",
      "full_name":"Любов Борисівна Федорчук",
      "position":"Student",
      "groupe_full_name":"IA-51",
      "faculty_name":"FICT",
      "department_name":"ACTS",
      "course":"4"
   },
   "error":{
      "value":"nil"
   }
}
```


Рисунок 4.11 — Об’єкт User у форматі JSON

Keychain — це спеціалізована база даних для зберігання метаданих та конфіденційної інформації безпосередньо на мобільному пристрої. Використання Keychain — це найкращий спосіб зберігати невеликі фрагменти даних, в цьому випадку, це пароль та електронна адреса. Взаємодія з Keychain реалізована в таких класах — KeychainPasswordItem, KeychainManager та KeychainConfiguration. Вихідний код наведений у додатку Б.5.

Основна задача цих класів — реалізація механізму збереження, читання та видалення даних користувача. Для цього реалізовано три функції в структурі KeychainPasswordItem: readPassword, savePassword, deleteItem.

– readPassword — функція для читання збережених користувацьких даних. Створюється запит, щоб знайти елемент, який відповідає сервісу, обліковому запису та групі доступу. Після цього вичитується збережений елемент ланцюжка ключів, який відповідає створеному запиту. Проводиться перевірка статус повернення й виводиться помилка, якщо це необхідно. Далі передається для використання рядок користувацьких даних з результату запиту.

– savePassword — функція для збереження даних користувача, яка приймає пароль користувача, в даному випадку це ключ, який створений шляхом поєднання електронної адреси користувача та паролю користувача, як параметр типу даних String. Спершу відбувається перевірка чи наявний такий пароль в базі. Для цього викликається функція readPassword, яка повертає наявний пароль, якщо він збережений. Після цього здійснюється перевірка на те, чи однакові існуючий та новий пароль. Якщо перевірка пройшла успішно й паролі виявилися різними, тоді старий пароль заміняється на новий. Якщо паролі однакові, тоді збереження не відбувається, так як такий пароль вже збережений і застосунок продовжує роботу в нормальному режимі.

– deleteItem — функція видалення з шифрованої бази Keychain користувацьких даних. Створюється запит для знайдення відповідного елементу для видалення, після чого знайдений елемент ланцюжка ключів видаляється. Проводиться перевірка статус повернення й виводиться помилка, якщо це необхідно.

Невід’ємною функціональною частиною мобільного застосунку є сповіщення користувача про збої та некоректну роботу застосунку. Приклад сповіщення показано на рисунку 4.12. Обробка помилок серверної та

клієнтської частин та відправка відповідних сповіщень реалізовані в класі Alert, вихідний код якого наведено в додатку Б.6.

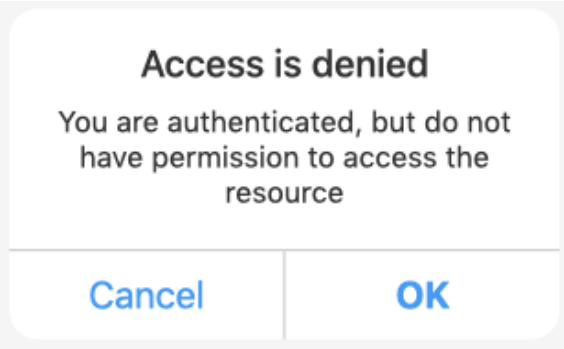


Рисунок 4.12 — Сповіщення користувачу про відмову у доступі

В класі реалізовано дві функції showAlertAccordingToStatusCode — функція, що відображає сповіщення користувачу в залежності від HTTP відповіді сервера та showAlert — функція, що відображає сповіщення про помилки в загальному вигляді.

4.4.2 Розробка компоненту розкладу занять

Наступним є компонент розкладу занять, основне призначення якого — це надання актуального розкладу занять користувачу, як студенту так і викладачу. Основною є робота з прикладним програмним інтерфейсом (англ. Application Programming Interface, API) сайту розкладу навчального закладу, в даному випадку, це — rozklad.org.ua. Даний сайт надає відкритий та документований API з можливістю виконання запитів по двох протоколах: HTTP або HTTPS та форматом даних відповіді на всі запити — JSON. Даний компонент розроблено за схожим до компоненту авторизації принципом. У класі TimetableService розроблено функціонал надсилання запиту до API сайту розкладу. Даний клас містить дві основні функції:

– getTimetableByGroupeName — функція для надсилання запиту на отримання розкладу занять по назві групи, де назва групи передається як параметр функції;

– getTimetableByTeacherName — функція для надсилання запиту на отримання розкладу занять викладача по імені.

У відповідь на запит, що надсилає клієнтський мобільний застосунок надходить список даних у форматі JSON, що відображається як масив об’єктів моделі, що реалізовано у вигляді класу Timetable. Даний клас містить всі необхідні поля такі як: номер поточного тижня, назва дня тижня, назва предмету, повне ім’я викладача, час початку та закінчення заняття, номер аудиторії. Відображення розкладу групи або викладача у вигляді списку реалізовано в класі TimetableTableViewController, що наслідується від відкритого класу UITableViewController, який надає механізм для роботи з елементом користувацького інтерфейсу таблиця.

Клас TimetableTableViewController надає можливість перегляду загального розкладу групи чи викладача на поточному тижні. В класі TimetableDetailsViewController, що наслідується від класу UIViewController розроблено функціонал перегляду деталей обраного зі списку заняття. Також реалізовано перехід на наступне представлення, де відображено деталі про викладача або групу, в залежності від того, який тип користувача наразі працює із застосунком. Вихідний код компоненту розкладу занять наведено у додатку Б.7.

Особливістю даного компоненту є робота з фреймворком CoreData. CoreData — це фреймворк для зберігання та управління об’єктним графом моделі даних. CoreData абстрагує деталі відображення об’єктів в сховище, що значно спрощує збереження даних за допомогою лише мови програмування Swift без необхідності в адмініструванні бази даних. Схематичне зображення застосунку з використанням CoreData показано на рисунок 4.13.

Автономний режим роботи даного компоненту реалізований з використанням фреймворку CoreData, що імпортується в клас TimetableTableViewController та надає доступ до взаємодії з локальним сховищем даних.

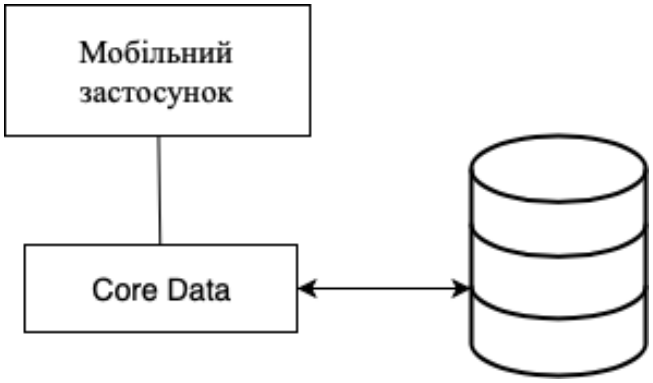


Рисунок 4.13 — Схема мобільного застосунку з CoreData

Основними елементами роботи з CoreData є:

– persistentContainer (координатор постійного сховища) — це прошарок між сховищем даних на мобільному застосунку та контекстом, в яких ці дані використовуються. Він відповідає за збреження, кешування користувацьких даних. На основі об'єктної керованої моделі створюється координатор постійного сховища та визначається, де саме повинні зберігатися дані та підключається сховище;

– managedObjectModel (керована об'єктна модель) — це модель даних, що містить всі сутності та взаємозв’язки між ними;

– saveContext — допоміжна функція для збереження даних. Збереження відбувається тільки в тому випадку, коли дані дійсно були оновлені.

За допомогою редактору була створена модель даних — сутність, або таблиця в БД, якщо проводити паралель з СУБД, Timetables, яка містить атрибути (поля таблиці) — назва предмету, номер аудиторії, тип предмету, ім’я викладача та час початку і закінчення заняття. При завантаженні відповідного представлення з розкладом заняття, в першу чергу,

перевіряється наявність підключення до мережі інтернет. У випадку, коли підключення відсутнє та даних наразі не було збережено в БД, тоді користувачу прийде сповіщення, про необхідність підключення до мережі для завантаження даних.

Якщо мобільний пристрій підключено до мережі, тоді дані будуть завантаженні за допомогою функції getTimetableByGroupeName та getTimetableByTeacherName. Після отримання даних, якщо запит пройшов успішно, дані будуть збережені до сутності Timetables за допомогою функції saveDataToTimetableEntity, де параметрами є атрибути сутності. Спочатку з використанням persistentContainer підключається сховище для збереження, після чого створюється новий об’єкт сутності Timetables в який записуються отримані атрибути. Якщо на одному з етапів роботи з БД виникли помилки, то з використанням механізму обробки помилок вони будуть перехоплені та оброблені відповідним чином. Запит на оновлення даних може бути ініційований користувачем або автоматично при кожному завантаженні представлення з розкладом занять.

4.4.3 Розробка компоненту управління календарем

Одним з компонентів мобільного застосунку взаємодії викладача та студента є календар. Робота з календарем реалізована з використанням бібліотеки JTAppleCalendar. Основною перевагою даної бібліотеки є що це бібліотека з відкритим вихідним кодом. Бібліотека надає широкі функціональні можливості з налаштування всіх необхідних для роботи елементів. Дана бібліотека нічого не проектує, лише надає макет з яким надалі відбувається робота. Представлення календаря містить два об’єкти, робота з якими реалізована в класі CalendarViewController.

Даний клас наслідує відкритий клас UIViewController та реалізує протоколи UITableViewDelegate й UITableViewDataSource — для роботи зі списком створених подій в календарі. Також реалізовано розширення класу

CalendarViewController. Розширення надають нові функціональні можливості до вже існуючих класів, структур, переліків або протоколів. Це дає можливість розширювати типи, до вихідного коду яких немає доступу. В даному випадку створено розширення для роботи з календарем за допомогою наслідування від протоколу JTAppleCalendarViewDataSource. Розроблено такі основні функції для роботи з календарем — configureCalendar, що показана на рисунку 4.14 та handleCellEvents — для роботи з обраною коміркою календаря по заданій даті.

```
extension CalendarViewController: JTAppleCalendarViewDataSource {
    func configureCalendar(_ calendar: JTAppleCalendarView) -> ConfigurationParameters {
        let formatter = DateFormatter()
        formatter.dateFormat = "yyyy MM dd"
        let startDate = Date()
        let calendar = Calendar.current
        let endDate = calendar.date(byAdding: .year, value: 1, to: Date())!
        return ConfigurationParameters(startDate: startDate, endDate: endDate)
    }
```

Рисунок 4.14 — Функція configureCalendar

Перегляд обраної зі списку події реалізовано в класі EventDetailsViewController та створення нової події, що додається в список запланованих подій відповідно до введеного дня та часу події реалізовано в класі AddEventViewController. Створена подія відображається по обраному дню в календарі. Якщо на обраний день заплановано декілька подій — відображення буде реалізовано у вигляді інтерактивного списку.

4.4.4 Розробка компоненту керування файловим сховищем

Ключовим компонентом в розробленому модулі є компонент файлового сховища. Він розроблений за принципами файлового сховища Dropbox. В даному компоненті реалізовано основний функціонал для роботи з файловим сховищем. Основною задачею даного компоненту була

можливість перегляду необхідних навчальних та методичних матеріалів в автономному режимі, тобто при відсутності підключення до мережі. Також невід'ємною функціональною частиною є надання доступу до останньої версії оновлених даних та синхронізації.

Всі зміни та маніпуляції, які проводилися з файлами поки мобільний пристрій не був підключений до мережі зберігаються на пристрої. Коли користувач наступного разу скористається даним компонентом, тобто перейде до екрану перегляду файлів, тоді оновлені дані будуть відправлені на сервер. Цей функціонал реалізовано у класі, що відповідає за взаємодію по мережі за допомогою протоколу HTTP — FileService. В цьому класі реалізовано чотири функції:

– getAllFolders — функція для надсилання GET запиту на отримання всіх папок, після чого ці дані відображаються користувачу у вигляді колекції комірок відсортовані по алфавіту за предметом та типом заняття, тобто лекційне, практичне або лабораторне заняття;

– getAllFilesByFolderTitle — функція для відправки GET запиту на отримання вмісту папки за її назвою, що відображаються користувачу у вигляді інтерактивного списку;

– updateFile — функція, що реалізує функціонал для відправки оновлених даних з клієнтського застосунку. Дані, що були оновлені переформовуються у JSON формат даних та надсилаються параметром у запиті. В даному випадку використовується метод PUT, основна задача якого завантаження даних, що передаються у запиті на вказаний ресурс. Якщо дані на вказаному ресурсі вже існують, то вони оновлюються, якщо ще не існують, тоді відбувається створення даних на вказаному ресурсі;

– createNewFile — функція для надсилання даних, створених на клієнтському застосунку, на вказаний ресурс. Для цього був використаний HTTP метод POST, який застосовується для відправки користувацьких даних заданому ресурсу. Створений об'єкт File відображається у формат даних JSON, який передається параметром в тілі запиту на визначений ресурс.

Якщо запит виконано вдало, то на клієнт надходить відповідь, що містить код стану 200ОК, що вказує на успішність запиту, та об’єкт для відображення в списку представлення застосунку. Вихідний код вищеописаних функцій наведено у додатку Б.8.

Для зручності у використанні створено окреме представлення, тобто екран на якому відображаються відсортовані комірки, що працюють за принципом папок у файловій системи. Функціонал роботи з комірками колекції, що містять назву предмету та тип заняття реалізовано в класі FoldersTableViewController.

Функціонал перегляду вмісту комірок, тобто переліку файлів, що відповідають предмету та типу заняття, реалізовано в класі DocumentsTableViewController. Для втілення автономного перегляду документів було використано фреймворк CoreData. Для цього було створено дві сутності Files та Folders, які мають атрибути назва та опис. Між двома сутностями було створено зв’язок один до багатьох, так як отримані файли мають залежали від комірки до якої вони належать відповідно до типу заняття та предмета. Для первинного використання компоненту керування файловим сховищем мобільний пристрій має бути підключеним до мережі, для завантаження всіх даних, а саме файлів, до застосунку. Це реалізується за допомогою вищеописаних функцій getAllFolders та getAllFilesByFolderTitle. Якщо пристрій не підключений до мережі інтернет, то користувачу буде приходити сповіщення про необхідність під’єднання мобільного пристрою до мережі. Збереження отриманих даних реалізовано за допомогою функцій saveFolderDataToCoreData та saveFilesToCoreData, що за поведінкою схожі на SQL запити, проте розроблені з використанням можливостей мови програмування Swift. При кожному завантажені даного компоненту буде ініційовано запит на оновлення даних за допомогою функції updateFile, що описана вище. Також користувач, якщо вважає необхідним, має можливість власноруч ініціювати оновлення даних. Цей механізм реалізовано функцією refreshDocuments.

Механізм видалення даних реалізовано за допомогою функції deleteDataFromCoreData. Розроблений механізм працює наступним чином:
Для роботи з CoreData необхідно створити об’єкт persistentContainer, що виконує роль прошарку між сховищем даних та контекстом. Після цього для виконання пошуку даних у сховищі створюється контекст з даного контейнера. Для видалення даних зі сховища, спершу, необхідно виконали пошук об’єкта, який необхідно видалити за допомогою fetchRequest. Після чого необхідно створити запит з предикатом для сутності, в даному випадку це Files та вилучити запис, який користувач бажає видалити. Особливістю видалення даних на відміну від оновлення та збереження є те, що видаляються вони тільки з клієнтського застосунку, запит на видалення даних з сервера буде надісланий лише в тому випадку, коли операцію видалення виконає останній користувач, що має доступ до цих даних. Цією дією буде ініційовано запит на оновлення даних на віддаленому ресурсі. У тілі відповіді на HTTP запит по оновленню даних буде надіслано маркування, що останній користувач видалив дані й після цього буде викликана функція deleteFile, що реалізує HTTP метод — DELETE. Після цього дані будуть остаточно видалені.

4.4.5 Розробка компоненту обліку академічної успішності

Одним з основних компонентів мобільного застосунку взаємодії викладача та студента є компонент обліку академічної успішності. Даний компонент розроблений за принципами побудови застосунку Google Sheets та електронного кампусу КПІ. Основна задача даного компоненту зі сторони студента — перегляд власної академічної успішності для ведення самоконтролю. З боку студента надається персональний доступ до власної успішності з кожного предмету в поточному навчальному році. Дані студенту доступні тільки для перегляду, внесення змін з власного облікового запису не передбачено. Так само студент не має доступу до успішності інших

студентів групи, так як це вважається персональною інформацією, яку не доцільно розголошувати без згоди інших користувачів.

Функціонал для перегляду академічної успішності студента реалізовано в класах AcademicProgressTableViewController та AcademicProgressDetailsTableViewController. Запити на завантаження успішності студента реалізовано в класі AcademicProgressService, за допомогою двох функцій:

– getSubjects — функція для отримання списку предметів для поточного користувача в поточному навчальному році, де параметром передається назва групи для пошуку списку предметів;

– getAcademicProgressByStudentName — функція, що реалізує запит на отримання академічної успішності поточного користувача з обраного предмету.

Ключовою задачею компоненту обліку академічної успішності студента з боку викладача є ведення та облік академічної успішності студентів груп, у яких даний викладач проводить заняття будь якого типу. За схожим принципом викладач має можливість переглядати список груп та предметів, які він викладає. Цей функціонал реалізовано у класах AcademicProgressTableViewController та у AcademicProgressService за допомогою функцій, що реалізують запити на отримання відповідних списків для відображення на представленні — getSubjectsList, де параметром передається повне ім’я викладача. Так як основна задача даного компоненту — це облік успішності, то для цього реалізовано функціонал проставлення відвідувань занять та відповідних відміток, балів, отриманих на занятті. Це реалізовано за допомогою таких функцій:

– updateAcademicProgress — функція з надсилання HTTP запиту PUT по оновленню даних;

– getAcademicProgressForGroupe — функція, що реалізує надсилання запиту на завантаження поточного списку групи з конкретного предмету для поточного користувача, для перегляду або внесенню змін.

У викладача не передбачено функціонал створення нових таблиць з обліку успішності з міркувань безпеки. Особливо наголошується на тому, що під час використання даного компоненту пристрій користувача має бути підключеним до мережі інтернет для коректної та безперебійної роботи застосунку.

4.4.6 Розробка компоненту обміну миттєвими повідомленнями

Однією з основних задач дипломного проекту було виокремлення єдиного засобу взаємодії викладача та студента невід'ємною складовою якого в сучасному світі є обмін миттєвими повідомленнями. Функціонал даного компоненту розроблювався на основуючись на описаному в розділі 2 сервісі Telegram. Як було зазначено у розділі 2, Telegram є найкращим сервісу для обміну миттєвими повідомленнями. Саме тому при розробці даного компоненту були враховані всі переваги застосунку Telegram. Для реалізації механізму обміну миттєвими повідомленнями в реальному часі була використана хмарна база даних Cloud Firestore, а для налаштування звичного користувацького інтерфейсу для обміну повідомленнями була імпортована бібліотека MessageKit. Дана бібліотека використана по причині того, що в мобільній ОС iOS не передбачено надання доступу до стандартних елементів UI, що вимагаються стандартом, який було згадано у розділі 3, Human Interface Guidelines.

Cloud Firestore — це хмарна NoSQL БД, яка використовується для збереження та синхронізації даних для розробки на стороні клієнта [19]. Перевагами такої хмарної бази даних є те, що вона легко масштабована, гнучка та підтримує синхронізацію даних в режимі реального часу. Дана база даних надається до використання безкоштовно, легко інтегрується з іншими продуктами Firebase та Google Cloud Platform, що в майбутньому можуть бути використані для розширення функціональності даного застосунку. Гнучкість зумовлюється тим, що модель даних в Cloud Firestore підтримує

ієрархічні структури даних, що організовано в колекції, які можуть мати глибоку вкладеність. Дана БД підтримує синхронізацію даних на будь-якому підключеному пристрої, також може бути ефективно використана для простих одноразових запитів, так як запити індексуються за замовчуванням, тому швидкодія запитів пропорційна розміру результуючого набору, а не набору даних. Однією з ключових переваг використання даної БД є підтримка автономного режиму. Cloud Firestore автоматично кешує дані, що активно використовуються, вони стають доступними для запису та читання. Оновлення даних відбувається після повернення пристрою в оперативний режим, тоді локальні дані синхронізуються з Cloud Firestore. Враховуючи те, що в компоненті використовуються неструктуровані дані, то використання NoSQL БД є цілком доцільним та обґрунтованим.

NoSQL БД — це база даних, яка забезпечує збереження та читання даних, використовуючи відмінний підхід, що застосовується в реляційних БД, що зумовлюється можливістю збереження неструктурованих даних, що можуть мати різну глибину вкладеності [20]. Cloud Firestore підтримує денормалізовану структуру даних, але в майбутньому це може стати значним недоліком. Денормалізована структура даних означає, що велика кількість даних буде дублюватися. З нарощуванням кількості користувачів це може стати проблемою, так як користувацькі дані будуть потребувати багато ресурсів. Хоча використовуючи такий підхід пошук даних буде більш швидким.

В першу чергу була створена структура даних. Так як у всьому проекті використаний формат даних JSON, то саме в цьому форматі доцільно зберігати дані в БД, що є можливим через те, що Cloud Firestore це NoSQL БД. Приклад об’єкту у форматі JSON наведено на рисунку 4.15.

Для роботи з хмарним сховищем було імпортовано модулі Firebase й FirebaseFirestore у класи компоненту обміну миттєвими повідомленнями та створено файл конфігурації сховища, що показаний на рисунку 4.16.

```
{
  "channels": [{
    "MOuL1sdbrnh0x1zGuXn7": {
      "name": “IA-51 chat“,
      "thread": [{
        "3a6Fo5rrUcBqhUJcLsP0": {
          "content": “Повідомлення 2",
          "created": "May 27, 2019 at 12:44:11 PM UTC-5",
          "senderID": "YCrPJF3shzWSHagmr0Zl2WZFBgT2",
          "senderName": “Староста IA-51“,
        },
        "4LXlVnWnoqyZEuKiiubh": {
          "content": “Повідомлення 1",
          "created": "May 27, 2018 at 12:40:05 PM UTC-5",
          "senderID": "f84PFeGl2yaqUDaSiTVeqe9gHfD3",
          "senderName": “Федорчук Любов Борисівна",
        },
      }]
    },
  }]
}
```


Рисунок 4.15 — Об’єкт даних повідомлення у форматі JSON


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>CLIENT_ID</key>
	<string>344283362860-
tqktp2dtvukb9883975kr1l1pr7bn1q4.apps.googleusercontent.com</string>
	<key>REVERSED_CLIENT_ID</key>
	<string>com.googleusercontent.apps.344283362860-
tqktp2dtvukb9883975kr1l1pr7bn1q4</string>
	<key>API_KEY</key>
	<string>AIzaSyAR2CCkpGZSn_f19YIYLfoAfK9Jnpms-ok</string>
	<key>GCM_SENDER_ID</key>
	<string>344283362860</string>
	<key>PLIST_VERSION</key>
	<string>1</string>
	<key>BUNDLE_ID</key>
	<string>com.fedorchuk.ios-client-campus</string>
	<key>PROJECT_ID</key>
	<string>ios-client-campus</string>
	<key>STORAGE_BUCKET</key>
	<string>ios-client-campus.appspot.com</string>
	<key>IS_ADS_ENABLED</key>
	<false></false>
	<key>IS_ANALYTICS_ENABLED</key>
	<false></false>
	<key>IS_APPINVITE_ENABLED</key>
	<true></true>
	<key>IS_GCM_ENABLED</key>
	<true></true>
	<key>IS_SIGNIN_ENABLED</key>
	<true></true>
	<key>GOOGLE_APP_ID</key>
	<string>1:344283362860:ios:79fd6e721b674fca</string>
	<key>DATABASE_URL</key>
	<string>https://ios-client-campus.firebaseio.com</string>
</dict>
</plist>
```


Рисунок 4.16 — Файл конфігурації хмарного сховища

Весь функціонал взаємодії користувача з сервісом обміну повідомлень реалізовано в класі ChatsTableViewController. При завантажені представлення сервісу обміну повідомленнями або безпосередньо чату викликаються функції, що відповідають за налаштування інтерфейсу компоненту. Це реалізовано шляхом реалізації протоколів, які надає бібліотека MessageKit — MessagesDisplayDelegate, MessagesLayoutDelegate та MessagesDataSource. Також реалізовано функціонал керування сервісом для обміну миттєвими повідомленнями. Це зроблено за допомогою функцій:

– createChannel — функція створення нового сервісу для обміну повідомленнями;

– addChannelToTable — функція для додавання нового сервісу в список чатів;

– updateChannelInTable — функція оновлення сервісу, в якій реалізовано сортування списку сервісів обміну повідомленнями по часу оновлення;

– removeChannelFromTable — функція видалення сервісу зі списку.

Функціонал обміну повідомленнями між користувачами реалізовано в класі ChatDetailsViewController. При ініціалізації об’єкту класу ChatDetailsViewController створюються об’єкти моделі даних User, Channel та масив об’єктів Message.

Також ініціалізується екземпляр Cloud Firestore, ListenerRegistration — основна задача якого виконувати очистку, коли це необхідно й CollectionReference — це точка в БД, де зберігаються повідомлення користувачів. Також Firestore викликає прослуховувач знімка щоразу, коли відбувається зміна в БД.

Основні функції для обробки повідомлень наведені нижче:

– insertNewMessage — функція, що перевіряє, що масив повідомлень ще не містить повідомлення, а потім додає його в представлення колекції;

– saveNewMessage — функція для збереження даних в БД, що використовує посилання на БД, що було попередньо налаштовано;

– handleDocumentChange — функція, що стежить за новими змінами в БД.

Варто зазначити, що даний компонент працює в режимі реального часу. Це забезпечується використанням хмарної БД реального часу, функціонал якої впроваджено за допомогою використання Firestore SDK. Інакше кажучи, за допомогою відкритого з'єднання та активного прослуховувача, як тільки один із користувачів компоненту зумовлює зміни в БД, тобто надсилає повідомлення, в той же момент прослуховувач реагує на зміни та викликає функцію для збереження та синхронізації даних чату.

4.4.7 Розробка допоміжних компонентів застосунку

Як було визначено у розділі 1, даний мобільний застосунок — це застосунок гібридного типу. Це зумовлено використанням вбудованого браузера у застосунку. Особливістю компоненту перегляду новин є робота з браузером. В тілі відповіді на GET запит клієнтського застосунку надходить посилання на обрану статтю з серверної частини. Обравши елемент зі списку, взаємодія з яким реалізована в класі NewsTableViewController так само як і у вищеописаних компонентах, відкривається вбудований браузер, що завантажує отримане посилання. Для роботи з браузером імпортується бібліотека WebKit.

WebKit — це ядро браузера, який використовується в браузері Apple Safari та надає обмежену функціональність, проте достатню для перегляду ряду посилань. WebKit пропонує класів для відображення веб-контенту у вікнах й реалізує такі функції браузера, як перехід по посиланнях при натисканні користувачем на них, керування історією недавно відвіданих сторінок та дозволяє переходити з початкового посилання в глибину використовуючи одне й те саме вікно. WebKit також спрощує процес

відображення контенту користувачу, який може містити різні типи контенту сторінок.

Ще одним допоміжним компонентом є перегляд облікового запису користувача, де вказана вся необхідна інформація, а саме ім’я та прізвище, номер телефону, електронна адреса, факультет, кафедра, позиція та міститься посилання на представлення розкладу занять. Особливим чином необхідно виділити допоміжний компонент налаштування, де користувач має можливість зміни власних даних, а саме номеру телефону, електронної адреси та паролю від облікового запису. Після зміни користувачем його паролю, викликається функція видалення та збереження нового паролю до шифрованого сховища Keychain. Після виконання цих операцій у користувача немає необхідності повторно авторизуватися, так як його дані були оновлені локально.

Для наведення статичної структури моделі проекту було розроблено діаграму класів, яка відображає взаємозв’язки між окремими сутностями та компонентами проекту, описує їх внутрішню структуру та типи відносин між ними. Діаграми класів застосунку взаємодії викладач-студент наведено на кресленнях Д3, Д4, Д5.

5 РОЗРОБКА ІНСТРУКЦІЇ КОРИСТУВАЧА

Мобільний застосунок розроблено на мові програмування Swift для ОС iOS, що дає можливість експлуатувати на мобільних пристроях під управлінням ОС iOS. Встановлені особливі вимоги до версії ОС — для коректної роботи застосунку на пристрої має бути встановлена версія не нижче 12.2.

Таким чином, мобільний застосунок можу бути встановлений на пристрої, що випущені компанією Apple моделі не раніше ніж — iPhone 5S. Мова Swift підтримує зворотню сумісність, тому мобільний застосунок буде працювати в режимі обмеженої функціональності на пристроях на яких встановлена версія ОС iOS не нижче 10.3.3. Це означає, що в такому режимі мобільний застосунок може працювати на пристроях моделі не раніше ніж — iPhone 5C.

На даному етапі мобільний застосунок можна завантажити на мобільний пристрій лише виконавши таку послідовність дій:

1. Завантажити вихідний код мобільного застосунку взаємодії викладач-студент. Вихідний код застосунку розміщено у вільному доступі на найбільшому веб-сервісі для хостингу ІТ-проектів GitHub. Завантажити можна напряму на власний ПК шляхом використання веб-інтерфейсу сервісу GitHub або скористатися системою контролю версій Git, яку необхідно попередньо встановити на ПК;
2. Після завантаження вихідного коду проекту необхідно завантажити на ПК засіб розробки Xcode або AppCode;
3. Наступним кроком буде запуск проекту мобільного застосунку взаємодії викладач-студент використовуючи один з попередньо встановлених засобів розробки;
4. Далі необхідно під’єднати власний мобільний пристрій з ОС iOS до ПК, після чого обраний засіб розробки розпізнає під’єднаний пристрій;

5. Після цього необхідно запустити проект натиснувши відповідну кнопку Build&Run.

6. Так як запуск мобільного застосунку відбувається на пристрої вперше на екран ПК буде надіслано сповіщення з інструкцією по конфігурації мобільного пристрою;

7. Останнім кроком буде повторний запуск проекту після чого мобільний застосунок буде доступним до використання на мобільному пристрої.

Під час первинного запуску мобільного застосунку необхідно, щоб пристрій був підключений до мережі інтернет для початкової авторизації користувача та завантаження даних на мобільний пристрій. При подальшому використанні застосунку підключення до мережі не є обов’язковим, так як мобільний застосунок може працювати в автономному режимі.

Однією з основних вимог до мобільного застосунку взаємодії викладач-студент була розробка гнучкого та інтуїтивно зрозумілого користувацького інтерфейсу. Про це свідчить скрішнот мобільного застосунку, що показаний на рисунку 4.17.

Про розробці інтерфейсу застосунку були враховані всі вимоги стандарту розробки під ОС iOS, що викладені як набір правил у Human Interface Guidelines. Для створення користувацького інтерфейсу були використані лише стандартні системні компоненти, такі як кнопки, панель вкладок, текстові поля, інтерактивні списки та колекції.

Так як дизайн мобільного застосунку створено звичним та інтуїтивно зрозумілим для користувача мобільного пристрою з ОС iOS, то відсутня необхідність в подальшому описі інтерфейсу.

Для детального опису всіх функціональних можливостей мобільного застосунку взаємодії викладач-студент розроблено діаграму варіантів використання. Діаграма варіантів використання описує взаємодію користувача з мобільним застосунком.

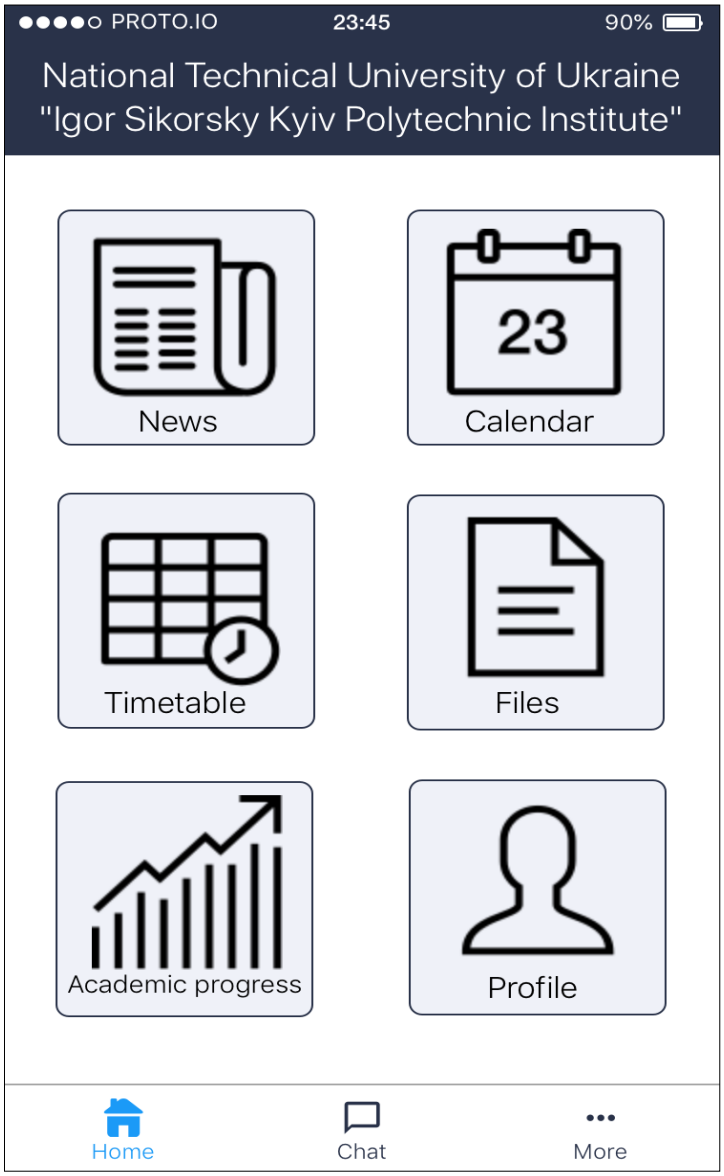


Рисунок 4.17 — Домашня сторінка мобільного застосунку взаємодії викладач-студент

Так як в проекті передбачено два типи дійових осіб, то було створено дві діаграми варіантів використання з боку актора студента та актора викладач. Діаграми варіантів використання наведено на кресленнях Д1 та Д2.

6 ПЕРСПЕКТИВИ РОЗШИРЕННЯ ЗАСТОСУНКУ ВЗАЄМОДІЇ ВИКЛАДАЧ-СТУДЕНТ

Ключовим моментом в перспективах розвитку мобільного застосунку для взаємодії викладача та студента є спрощення процесу доставки застосунку кінцевому користувачеві. Щоб полегшити цей процес необхідно опублікувати мобільний застосунок в App Store.

App Store — це платформа, що розроблена компанією Apple, для поширення мобільних застосунків для пристроїв з ОС iOS. Для публікації мобільного застосунку треба пройти декілька етапів.

1. Створити обліковий запис розробника. Для цього необхідно заповнити відповідну форму реєстрації, вказуючи дані розробника. Окремим аспектом є те, що створення облікового запису є платне. Після оформлення анкети та проведення оплати обліковий запис буде підтверджено впродовж двох тижнів;

2. Після цього необхідно підготувати всі матеріали для публікації. Створити текст опису мобільного застосунку відповідно до зазначених вимог, які викладені в інструкції Marketing Resources and Identity Guidelines. Опис застосунку має чітко та коротко доносити до користувача його основне призначення та функціональні можливості. Якщо застосунок потрапляє під вікові обмеження необхідно обов’язково їх зазначити в тексті. Найкращим рішенням буде вказати особливу функціональність застосунку та те, чим даний застосунок відрізняється від вже існуючих на ринку;

3. Окрім опису необхідно оформити від 3 до 5 скріншотів мобільного застосунку, що мають відповідати стандартам, наведеним у Marketing Resources and Identity Guidelines. Додатково вказуються також найменування правовласника, ключові слова для пошуку в магазині, відео-прев `ю та скріншоти для різних пристроїв;

4. Важливим моментом є забезпечення збірки наявністю сертифіката цифрового підпису;

5. Після підготовки всіх перелічених матеріалів наступним етапом буде слідувати покроковій інструкції по публікації, що вказана на iTunes Connect.

6. Після цього мобільний застосунок буде відправлено на розгляд та попереднє тестування, буде проводитись перевірка відповідності мобільного застосунку всім зазначеним стандартам. Впродовж двох тижнів мобільний застосунок буде опубліковано в App Store або відхилено із описом причин відмови в публікації. Після усунення виявлених помилок можливо знову відправити застосунок на розгляд до публікації.

Ще одним варіантом розширення застосунку взаємодії викладач студент є доповнення існуючого функціоналу. Для цього може бути створений ще один тип користувача — деканат. З впровадженням додаткового типу користувача зростають можливості нарощення функціональності мобільного застосунку. Стає можливим розробка функціоналу управління та обліку електронних відомостей. Процес створення електронних відомостей буде автоматизовано, буде забезпечено надійне на захищене зберігання та розмежування доступу до відомостей. Для користувача типу деканат необхідно дати доступ для перегляду статистики відвідування занять, автоматизувати підрахунок пропущених пар студентом та автоматичне надсилання попереджень щодо пропусків. Деканату буде надано доступ до надсилання розсилки з повідомленням важливої інформації, що безпосередньо стосується навчального процесу.

Окрім користувача деканат можливе створення особистого кабінету приймальної комісії, який полегшить та прискорить процес прийому документів абітурієнтів до навчального закладу. Також необхідно буде створення та адміністрування безпечного та надійного хмарного сховища для документів абітурієнтів навчального закладу.

ВИСНОВКИ

В ході виконання даного дипломного проекту було спроектовано та розроблено застосунок взаємодії викладач-студент.

Після аналізу предметної області та оцінки існуючих рішень було прийняте рішення про розробку мобільного застосунку саме під ОС iOS. Розроблений мобільний застосунок взаємодії викладач-студент повністю відповідає поставленим вимогам та виконує всі поставлені, в розділі 1, задачі. Для виконання поставлених задач були застосовані сучасні підходи та обрано прогресивні технології. Мобільний застосунок поєднує в собі функціональні можливості декількох сервісів та застосунків, що дозволяє скоротити кількість використовуваних застосунків, що в свою чергу, зберігає ресурси мобільних пристроїв — мобільний інтернет трафік, що використовується та енергетичні ресурси. Перевагами розробленого застосунку для взаємодії викладача та студента є:

– Забезпечення безперебійної роботи основних компонентів в автономному режимі, а саме — перегляд розкладу занять, доступ до навчальних матеріалів та останніх повідомлень в чаті;

– Надання функціональності обміну миттєвими повідомленнями у режимі реального часу;

– Забезпечення зворотної синхронізації користувацьких даних після повернення пристрою до оперативного режиму роботи;

– Надання можливості одноразової авторизації користувача зі збереженням користувацьких даних у шифрованому сховищі безпосередньо на мобільному пристрої.

– Надає можливість легкої підтримки та нарощування функціональності мобільного застосунку.

Недоліками розробленого мобільного застосунку є:

– Мобільний застосунок розроблено під ОС iOS, що на даний момент робить неможливим використання застосунку на пристроях з іншою мобільною ОС;

– Встановлення застосунку на мобільний пристрій не є тривіальною задачею для користувача та потребує виконання декількох описаних вище кроків;

– Відсутність можливості надсилання зображень, аудіо та відео записів з використанням компоненту обміну миттєвими повідомленнями;

– Застосунок розроблено з урахуванням особливостей та потреб навчального закладу КПІ ім. Ігоря Сікорського.

Завдяки використанню архітектурного підходу MVC зростає швидкість розробки та гнучкість розробленого застосунку, що надає можливості розширення та використання його іншими навчальними закладами, за умови внесення незначних змін у проект. Отже, виконано всі поставлені задачі з дотриманням всіх вимог до проектування та розробки мобільного застосунку під ОС iOS.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mobile App Usage - Statistics & Facts [Електронний ресурс]: Режим доступу: https://www.statista.com/topics/1002/mobile-app-usage/.
2. Annual number of global mobile app downloads 2017-2022 [Електронний ресурс]: Режим доступу: https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/
3. Mobile Marketing Statistics for 2019 and Beyond [Електронний ресурс]: Режим доступу: https://www.bluecorona.com/blog/mobile-marketing-statistics
4. Media, Productivity & Emojis Give Mobile Another Stunning Growth Year [Електронний ресурс]: Режим доступу: https://flurrymobile.tumblr.com/post/136677391508/stateofmobile2015#_=_
5. Електронний кампус НТУУ «КПІ імені Ігоря Сікорського» [Електронний ресурс]: Режим доступу: https://ecampus.kpi.ua/
6. Texas A&M University [Електронний ресурс]: Режим доступу: https://itunes.apple.com/us/app/texas-a-m-university/id318638320?mt=8
7. Dropbox [Електронний ресурс]: Режим доступу: https://www.dropbox.com/
8. Telegram [Електронний ресурс]: Режим доступу: https://telegram.org
9. Number of apps available in leading app stores 2019 [Електронний ресурс]: Режим доступу: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/
10. Platform versions [Електронний ресурс]: Режим доступу: https://developer.android.com/about/dashboards/index.html
11. Singh, Amit Mac OS X Internals: A Systems Approach – Addison-Wesley Professional, 2006. – 1641 с.
12. Swift.org [Електронний ресурс]: Режим доступу: https://swift.org
13. The State of Developer Ecosystem in 2018 [Електронний ресурс]: Режим доступу: https://www.jetbrains.com/research/devecosystem-2018/swift-objc/

14. Xcode [Електронний ресурс]: Режим доступу: https://developer.apple.com/xcode

15. Apache ActiveMQ [Електронний ресурс]: Режим доступу: http://activemq.apache.org

16. RabbitMQ is the most widely deployed open source message broker [Електронний ресурс]: Режим доступу: https://www.rabbitmq.com

17. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems [Електронний ресурс] Режим доступу: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems#database-management-systems

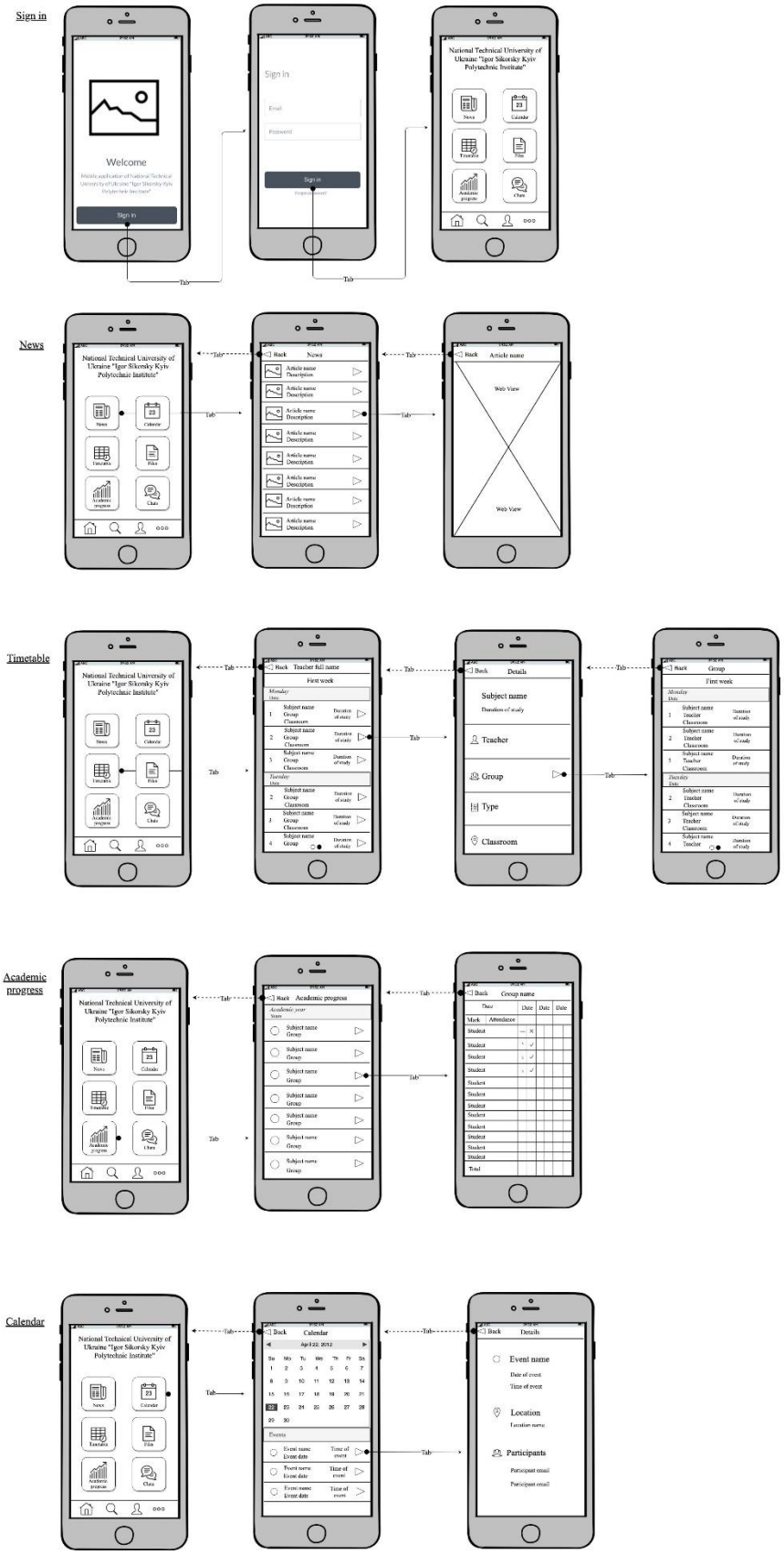
18. New data shows losing 80% of mobile users is normal, and why the best apps do better [Електронний ресурс] Режим доступу: https://andrewchen.co/new-data-shows-why-losing-80-of-your-mobile-users-is-normal-and-that-the-best-apps-do-much-better/

19. Firebase [Електронний ресурс] Режим доступу: https://firebase.google.com/

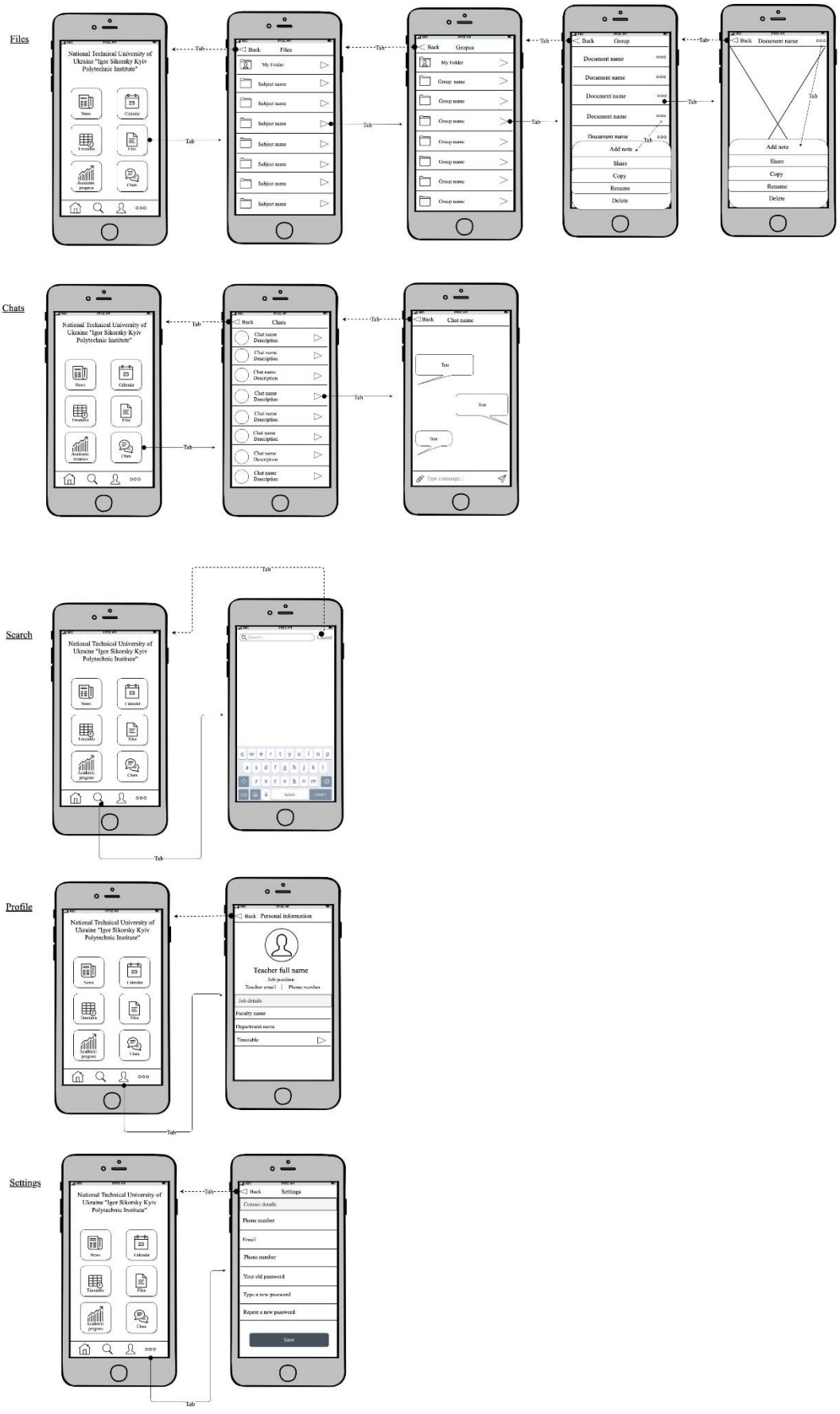
20. Martin Fowler NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence/ Martin Fowler, Pramodkumar J. Sadalage – Addison-Wesley Professional, 2012. – 190 с.

ДОДАТОК А

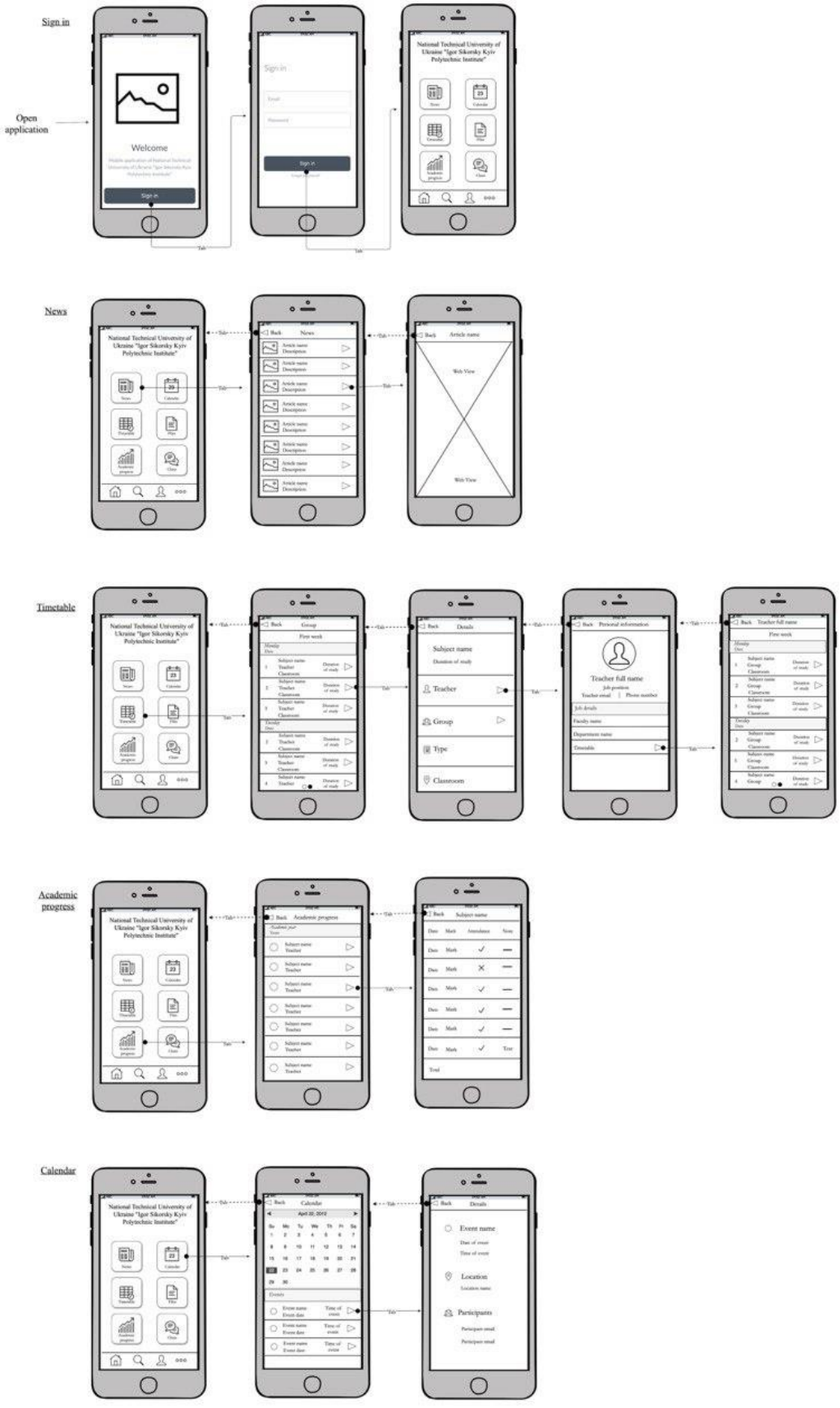
А.1 Прототип з боку користувача — викладач



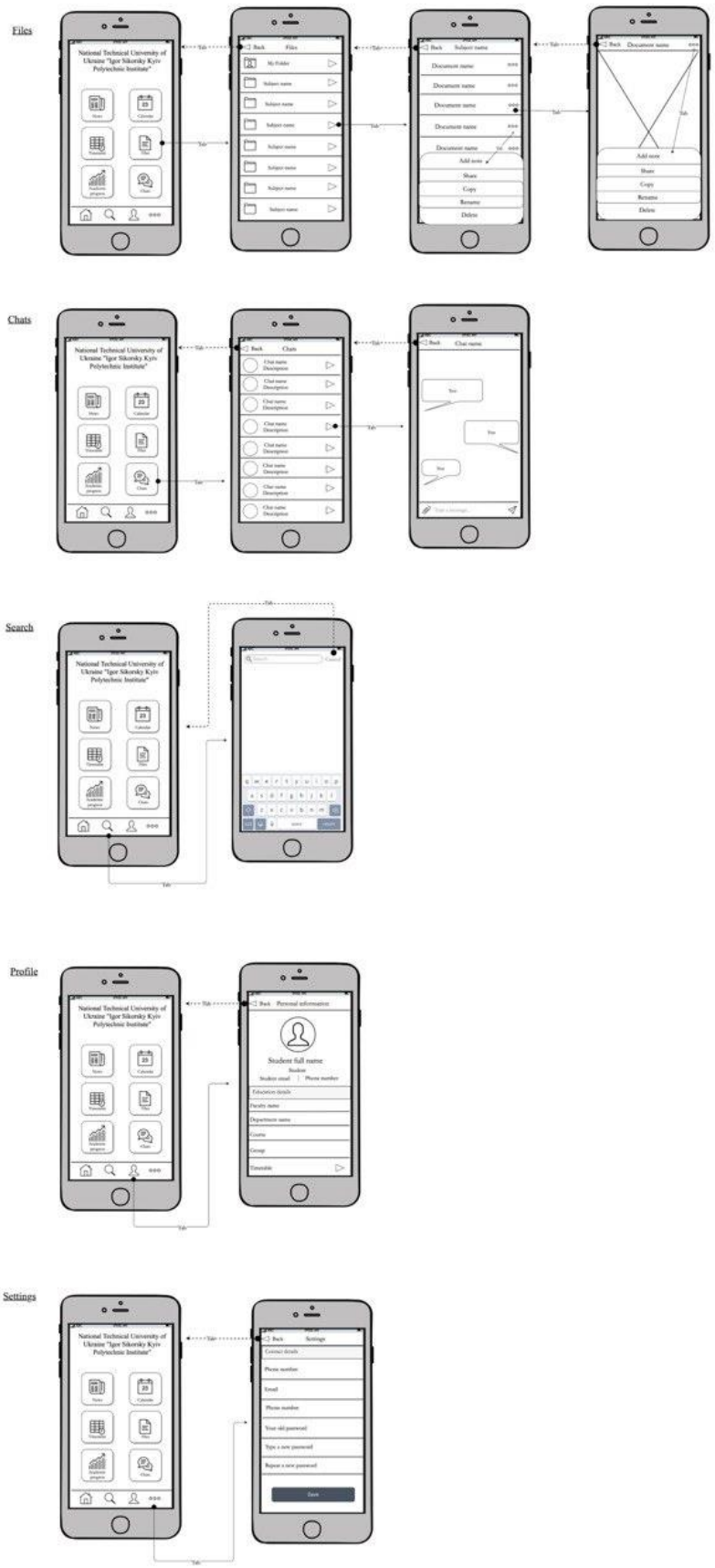
Продовження А.1



А.2 Прототип з боку користувача — студент



Продовження А.2



ДОДАТОК Б

Лістинг програми

Б.1 Вихідний код UserService

```
import Alamofire

class UserService {
   func getUser (headers: [String: String], completionHandler: @escaping (Int?) -> Void) {
      Alamofire.request(Constants.BASE_URL + "/users/current",
                        method: .get,
                        parameters: nil,
                        encoding: JSONEncoding.default,
                        headers: headers).responseJSON { response in
                         let status = response.response?.statusCode
                         switch response.result {
                         case .success:
                            guard let status = status else {
                               log.error("Request passed without status code - status code is nil.")
                               completionHandler(nil)
                               return
                            }

                            guard status == 200 else {
                               log.debug("Request passed with status code, but not 200 OK: \(status)")
                               completionHandler(status)
                               return
                            }

                            completionHandler(status)
                         case .failure(let error):
                            guard let status = status else {
                               log.debug("Request failure with error: \(error)")
                               completionHandler(nil)
                               return
                            }
                            log.debug("Request failure with status code: \(status)")
                            completionHandler(status)
                         }
      }
   }
}
```

Б.2 Вихідний код WelcomeViewController

```
import UIKit
class WelcomeViewController: UIViewController {
   @IBOutlet weak var logoImageView: UIImageView!
```

```
    @IBOutlet weak var welcomeLabel: UILabel!
    @IBOutlet weak var universityNameLabel: UILabel!
    @IBOutlet weak var signInButton: UIButton!

    let alert = Alert()
    override func viewDidLoad() {
        super.viewDidLoad()
        if !Connectivity.isConnectedToInternet {
            log.warning("No Internet Conection.Needed turning on cellular data or use Wifi to access
data.")
            let alertItem = alert.showAlert(alertTitle: "No Internet Conection", alertMessage: "Turn
on cellular data or use Wi-Fi to access data.")
            self.present(alertItem, animated: true, completion: nil)
            log.debug("Alert with no internet connection error presented successfully.")
        } else {
            let hasLogin = UserDefaults.standard.bool(forKey: "hasUserSecretAPIkey")
            if hasLogin {
                performSegue(withIdentifier: "dismissSignInView", sender: self)
                log.debug("User is log in.")
            } else {
                log.debug("User is not log in. Needed loginning.")
            }
            signInButtonSetUp(button: signInButton)
        }
    }

    func signInButtonSetUp(button: UIButton) {
        button.layer.cornerRadius = 7
        button.layer.borderWidth = 1
        button.layer.borderColor = UIColor(red: 48/255,
                                           green: 79/255,
                                           blue: 100/255,
                                           alpha: 1.0).cgColor
    }
}

import Alamofire

class Connectivity {
    class var isConnectedToInternet: Bool {
        return NetworkReachabilityManager()!.isReachable
    }
}
```

Б.3 Вихідний код SignInViewController

```
import UIKit
class SignInViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet weak var usernameTextField: UITextField!
```

```
    @IBOutlet weak var passwordTextField: UITextField!
    @IBOutlet weak var signInButton: UIButton!

    let alert = Alert()
    override func viewDidLoad() {
        super.viewDidLoad()
        if !Connectivity.isConnectedToInternet {
            log.warning("No Internet Conection.Needed turning on cellular data or use Wifi to access
data.")
            let alertItem = alert.showAlert(alertTitle: "No Internet Conection", alertMessage: "Turn
on cellular data or use Wi-Fi to access data.")
            self.present(alertItem, animated: true, completion: nil)
            log.debug("Alert with no internet connection error presented successfully.")
        } else {
            let hasLogin = UserDefaults.standard.bool(forKey: "hasUserSecretAPIkey")
            if hasLogin {
                performSegue(withIdentifier: "dismissSingInView", sender: self)
                log.debug("User is log in.")
            } else {
                log.debug("User is not log in. Needed loginning.")
            }
            self.usernameTextField.delegate = self
            self.passwordTextField.delegate = self
            signInButtonSetUp(button: signInButton)
        }
    }

    func signInButtonSetUp(button: UIButton) {
        button.layer.cornerRadius = 7
        button.layer.borderWidth = 1
        button.layer.borderColor = UIColor(red: 48/255,
                                           green: 79/255,
                                           blue: 100/255,
                                           alpha: 1.0).cgColor
    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        self.view.endEditing(true)
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
    }

    @IBAction func signInButtonTapped(_ sender: Any) {
        let newUsername = usernameTextField.text
        let newUserPassword = passwordTextField.text

        UserDefaults.standard.set(false, forKey: "hasUserSecretAPIkey")

        if (newUsername!.isEmpty && newUserPassword!.isEmpty) {
```

```
            let alertItem = self.alert.showAlert(alertTitle: "Text fields are empty",
                                                 alertMessage: "Please, enter your username and password.")
            self.present(alertItem, animated: true, completion: nil)
        } else if (newUsername?.isEmpty)! {
            let alertItem = self.alert.showAlert(alertTitle: "Text field is empty",
                                                 alertMessage: "Please, enter your username or email.")
            self.present(alertItem, animated: true, completion: nil)
        } else if (newUserPassword?.isEmpty)! {
            let alertItem = self.alert.showAlert(alertTitle: "Text field is empty",
                                                 alertMessage: "Please, enter your password.")
            self.present(alertItem, animated: true, completion: nil)
        } else {
            let newUserSecretAPIkey = newUsername! + newUserPassword!
            let keychainManager = KeychainManager()
            let userService = UserService()

            let headers = keychainManager.createAuthorizationHeadersForRequest(userApiKey:
newUserSecretAPIkey)
            userService.getUser(headers: headers, completionHandler: { status in
                guard let status = status else {
                    log.warning("Status code is nil.")
                    return
                }

                if status == 200 {
                    do {
                        let userSecretAPIKeyItem = KeychainPasswordItem(service:
KeychainConfiguration.serviceName, accessGroup: KeychainConfiguration.accessGroup)
                        try userSecretAPIKeyItem.savePassword(newUserSecretAPIkey)
                    } catch {
                        fatalError("Error updating keychain - \(error)")
                    }
                    UserDefaults.standard.set(true, forKey: "hasUserSecretAPIkey")
                    self.performSegue(withIdentifier: "dismissSignInView", sender: self)
                    log.debug("API key is valid")
                } else {
                    self.alert.showAlertAccordingToStatusCode(fromController: self, statusCode: status)
                }
            })
        }
    }
}
```

Б.4 Вихідний код моделі User

```
import Foundation

struct User: Decodable {
    let username: String
    let email: String
```

```
    let phone: String
    let fullName: String
    let position: String
    let faculty: String
    let department: String
    let course: Int?
    let groupe: String?

    private enum CodingKeys: String, CodingKey {
        case username
        case email
        case phone = "phone_number"
        case fullName = "full_name"
        case position
        case faculty = "faculty_name"
        case department = "department_name"
        case course
        case groupe = "group_full_name"
    }
}
```

Б.5 Вихідний код для роботи з Keychain

```
import Foundation
struct KeychainConfiguration {
    static let serviceName = "ios-client-campus"
    static let accessGroup: String? = nil
}

import Foundation
struct KeychainPasswordItem {
    enum KeychainError: Error {
        case noPassword
        case unexpectedPasswordData
        case unexpectedItemData
        case unhandledError(status: OSStatus)
    }

    let service: String
    let accessGroup: String?
    init(service: String,
         accessGroup: String? = nil) {
        self.service = service
        self.accessGroup = accessGroup
    }
    func readPassword() throws -> String  {
        var query = KeychainPasswordItem.keychainQuery(withService: service, accessGroup:
accessGroup)
        query[kSecMatchLimit as String] = kSecMatchLimitOne
        query[kSecReturnAttributes as String] = kCFBooleanTrue
```

```
        query[kSecReturnData as String] = kCFBooleanTrue
        var queryResult: AnyObject?
        let status = withUnsafeMutablePointer(to: &queryResult) {
            SecItemCopyMatching(query as CFDictionary, UnsafeMutablePointer($0))
        }
        guard status != errSecItemNotFound else { throw KeychainError.noPassword }
        guard status == noErr else { throw KeychainError.unhandledError(status: status) }
            guard let existingItem = queryResult as? [String : AnyObject],
            let passwordData = existingItem[kSecValueData as String] as? Data,
            let password = String(data: passwordData, encoding: String.Encoding.utf8)
            else {
                throw KeychainError.unexpectedPasswordData
        }

        return password
    }

    func savePassword(_ password: String) throws {
        let encodedPassword = password.data(using: String.Encoding.utf8)!

        do {
            try _ = readPassword()
            var attributesToUpdate = [String : AnyObject]()
            attributesToUpdate[kSecValueData as String] = encodedPassword as AnyObject?

            let query = KeychainPasswordItem.keychainQuery(withService: service, accessGroup:
accessGroup)
            let status = SecItemUpdate(query as CFDictionary, attributesToUpdate as CFDictionary)
            guard status == noErr else { throw KeychainError.unhandledError(status: status) }
        }
        catch KeychainError.noPassword {
            var newItem = KeychainPasswordItem.keychainQuery(withService: service,
accessGroup: accessGroup)
            newItem[kSecValueData as String] = encodedPassword as AnyObject?
            let status = SecItemAdd(newItem as CFDictionary, nil)
                    guard status == noErr else { throw KeychainError.unhandledError(status: status) }
        }
    }

    func deleteItem() throws {
        let query = KeychainPasswordItem.keychainQuery(withService: service, accessGroup:
accessGroup)
        let status = SecItemDelete(query as CFDictionary)
        guard status == noErr || status == errSecItemNotFound else { throw
KeychainError.unhandledError(status: status) }
    }

    static func passwordItems(forService service: String, accessGroup: String? = nil) throws ->
[KeychainPasswordItem] {
        var query = KeychainPasswordItem.keychainQuery(withService: service, accessGroup:
accessGroup)
        query[kSecMatchLimit as String] = kSecMatchLimitAll
```

```
        query[kSecReturnAttributes as String] = kCFBooleanTrue
        query[kSecReturnData as String] = kCFBooleanFalse
            var queryResult: AnyObject?
        let status = withUnsafeMutablePointer(to: &queryResult) {
            SecItemCopyMatching(query as CFDictionary, UnsafeMutablePointer($0))
        }

        guard status != errSecItemNotFound else { return [] }
        guard status == noErr else { throw KeychainError.unhandledError(status: status) }
            guard let resultData = queryResult as? [[String : AnyObject]] else { throw
KeychainError.unexpectedItemData }
            var passwordItems = [KeychainPasswordItem]()
        for _ in resultData {

            let passwordItem = KeychainPasswordItem(service: service, accessGroup: accessGroup)
            passwordItems.append(passwordItem)
        }

        return passwordItems
    }

    private static func keychainQuery(withService service: String, accessGroup: String? = nil) ->
[String : AnyObject] {
        var query = [String : AnyObject]()
        query[kSecClass as String] = kSecClassGenericPassword
        query[kSecAttrService as String] = service as AnyObject?

        if let accessGroup = accessGroup {
            query[kSecAttrAccessGroup as String] = accessGroup as AnyObject?
        }

        return query
    }
}

class KeychainManager {

    func deleteUserSecretAPIkeyFromKeychain() {
        do {
            let userSecretAPIKeyItem = KeychainPasswordItem(service:
KeychainConfiguration.serviceName,
                                                accessGroup: KeychainConfiguration.accessGroup)
            try userSecretAPIKeyItem.deleteItem()
            log.debug("Deleting keychain item from query is successful")
        }
        catch {
            log.error("Error deleting keychain item from query - \(error)")
            fatalError(error as! String)
        }
    }

    func readUserSecretAPIkeyFromKeychain() -> String {
```

```
        var userSecretAPIkey: String
        do {
            let userSecretAPIkeyItem = KeychainPasswordItem(service:
KeychainConfiguration.serviceName, accessGroup: KeychainConfiguration.accessGroup)
            userSecretAPIkey = try userSecretAPIkeyItem.readPassword()
        }
        catch {
            fatalError("Error reading secret API key from keychain - \(error)")
        }

        return userSecretAPIkey
    }

    func createAuthorizationHeadersForRequest(userApiKey: String?) -> [String: String] {
        guard let apiKey = userApiKey else {
            let userSecretAPIkeyUsingEncoding =
self.readUserSecretAPIkeyFromKeychain().data(using: String.Encoding.utf8)!
            let userSecretAPIkeyBase64Encoded =
userSecretAPIkeyUsingEncoding.base64EncodedString(options: [])
            let header = ["Authorization" : "Basic \(userSecretAPIkeyBase64Encoded)"]
            return header
        }
        let userSecretAPIkeyUsingEncoding = apiKey.data(using: String.Encoding.utf8)!
        let userSecretAPIkeyBase64Encoded =
userSecretAPIkeyUsingEncoding.base64EncodedString(options: [])
        let headers = ["Authorization" : "Basic \(userSecretAPIkeyBase64Encoded)"]
        return headers
    }
}
```

Б.6 Вихідний код Alert

```
import UIKit
class Alert {

    func showAlertAccordingToStatusCode(fromController controller: UIViewController,
                                        statusCode: Int) {
        switch(statusCode) {
        case 401:
            let alert = showAlert(alertTitle: "Invalid API key",
                                  alertMessage: "Your API key was incorrect. Please, check your API key
and enter it again.")
            controller.present(alert, animated: true, completion: nil)
            log.warning("Invalid API key")
        case 403:
            let alert = showAlert(alertTitle: "Access is denied",
                                  alertMessage: "You are authenticated, but do not have permission to access
the resource.")
            controller.present(alert, animated: true, completion: nil)
            log.warning("Access is denied")
```

```
        case 500...526:
            let alert = showAlert(alertTitle: "Service unavailable",
                                  alertMessage: "Please, try again later.")
            controller.present(alert, animated: true, completion: nil)
            log.warning("Service unavailable")
        default:
            let alert = showAlert(alertTitle: "Unexpected error",
                                  alertMessage: "Please, try again later.")
            controller.present(alert, animated: true, completion: nil)
            log.error("Unexpected error with status code: \(statusCode)")
        }
    }

    func showAlert(alertTitle: String,
                   alertMessage: String) -> UIAlertController {
        let alert = UIAlertController(title: alertTitle,
                                      message: alertMessage,
                                      preferredStyle: .alert)
        let okAction = UIAlertAction(title: "OK", style: UIAlertAction.Style.default) {
            UIAlertAction in
            UserDefaults.standard.set(true, forKey: "AcctionOkPressed")
            log.debug("Ok pressed")
        }
        alert.addAction(okAction)
        return alert
    }
}
```

Б.7 Вихідний код компоненту Timetable

```
import UIKit
import CoreData
class TimetableTableViewController: UITableViewController {

    var timetable: [NSManagedObject] = []
    let alert = Alert()
    var groupe: String?
    let userService = UserService()
    let timetableService = TimetableService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
    override func viewDidAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        self.tableView.rowHeight = 125

        guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else {
            return
        }
```

```
        let managedContext = appDelegate.persistentContainer.viewContext
        let fetchRequest = NSFetchRequest<NSManagedObject>(entityName: "Articles")
        do {
            timetable = try managedContext.fetch(fetchRequest)
            self.tableView.reloadData()
        } catch let error {
            log.error("Could not fetch. \(error), \(error.localizedDescription)")
        }
        if timetable.isEmpty {
            if !Connectivity.isConnectedToInternet {
                let alertItem = alert.showAlert(alertTitle: "No Internet Conection", alertMessage:
"Turn on cellural data or use Wi-Fi to access data.")
                self.present(alertItem, animated: true, completion: nil)
            } else {
                self.getUserGroupe()
                self.getTimetableByGroupeName(groupeName: groupe!)
            }
        }

        let refreshControl = UIRefreshControl()
        refreshControl.addTarget(self, action:  #selector(refresh), for:
UIControl.Event.valueChanged)
        tableView.refreshControl = refreshControl
    }

    @objc func refresh(refreshControl: UIRefreshControl) {
        self.getUserGroupe()
        self.getTimetableByGroupeName(groupeName: groupe!)
        tableView.reloadData()
        log.debug("Timetable has been refreshed.")
        DispatchQueue.main.asyncAfter(deadline: DispatchTime.now() + 2) {
            refreshControl.endRefreshing()
        }
    }

    func saveDataToTimetableEntity(lessonName: String, teacherName: String, lessonRoom:
String, lessonType: String, timeStart: String, timeEnd: String) {
        guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else {
            return
        }

        let managedContext = appDelegate.persistentContainer.viewContext
        let entity = NSEntityDescription.entity(forEntityName: "Timetable", in: managedContext)!
        let newTimetable = NSManagedObject(entity: entity, insertInto: managedContext)

        newTimetable.setValue(lessonName, forKeyPath: "lessonName")
        newTimetable.setValue(teacherName, forKey: "teacherName")
        newTimetable.setValue(lessonRoom, forKey: "lessonRoom")
        newTimetable.setValue(lessonType, forKey: "lessonType")
        newTimetable.setValue(timeStart, forKey: "timeStart")
        newTimetable.setValue(timeEnd, forKey: "timeEnd")
```

```
        do {
            try managedContext.save()
            timetable.append(newTimetable)
        } catch let error {
            log.error("Could not fetch. \(error), \(error.localizedDescription)")
        }
        tableView.reloadData()
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int {
        return timetable.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        guard let cell = tableView.dequeueReusableCell(withIdentifier: "timetableTableViewCell",
for: indexPath) as? TimetableTableViewCell else {
            fatalError("The dequeued cell is not an instance of TimetableTableViewCell.")
        }

        let timetableItem = timetable[indexPath.row]
        cell.lessonNameLabel.text = timetableItem.value(forKeyPath: "lessonName") as? String
        cell.teacherNameLabel.text = timetableItem.value(forKeyPath: "teacherName") as? String
        cell.classroomLabel.text = timetableItem.value(forKeyPath: "lessonRoom") as? String
        cell.typeLabel.text = timetableItem.value(forKeyPath: "lessonType") as? String
        return cell
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        super.prepare(for: segue, sender: sender)
        switch(segue.identifier ?? "") {
        case "showLessonDetails":
            guard let timetableDetailsViewController = segue.destination as?
TimetableDetailsViewController else {
                log.error("Unexpected destination: \(segue.destination)")
                fatalError("Unexpected destination: \(segue.destination)")
            }

            guard let selectedTimetableTableViewCell = sender as? TimetableTableViewCell else {
                log.error("Unexpected sender: \(String(describing: sender))")
                fatalError("Unexpected sender: \(String(describing: sender))")
            }

            guard let indexPath = tableView.indexPath(for: selectedTimetableTableViewCell) else {
                log.error("The selected cell is not being displayed by the table")
                fatalError("The selected cell is not being displayed by the table")
            }

            let selectedLesson = timetable[indexPath.row]
```

```
            timetableDetailsViewController.lessonName = selectedLesson.value(forKey:
"lessonName") as? String
            timetableDetailsViewController.classroom = selectedLesson.value(forKey:
"lessonRoom") as? String
            timetableDetailsViewController.teacherName = selectedLesson.value(forKey:
"teacherName") as? String
            timetableDetailsViewController.type = selectedLesson.value(forKey: "lessonType") as?
String
            timetableDetailsViewController.startTime = selectedLesson.value(forKey: "timeStart")
as? String
            timetableDetailsViewController.endTime = selectedLesson.value(forKey: "timeEnd") as?
String

        default:
            log.error("Unexpected Segue Identifier; \(String(describing: segue.identifier))")
            fatalError("Unexpected Segue Identifier; \(String(describing: segue.identifier))")
        }
    }

    private func getTimetableByGroupeName(groupeName: String) {
        timetableService.getTimetableByGroupeName(groupeName: groupeName,
completionHandler: { timetableItem, status in
            if (status != nil && status == 200) {
                guard let timetable = timetableItem,
                    let statusCode = status else {
                        let alertItem = self.alert.showAlert(alertTitle: "Unexpected error", alertMessage:
"Please, try again later.")
                        self.present(alertItem, animated: true, completion: nil)
                        log.error("Unexpected error without status code.")
                        return
                }

                if statusCode == 200 {
                    for timetableItem in timetable {
                        self.save(lessonName: timetableItem.lessonName,
                                  teacherName: timetableItem.lessonRoom,
                                  lessonRoom: timetableItem.teacherName,
                                  lessonType: timetableItem.lessonType,
                                  timeStart: timetableItem.timeStart,
                                  timeEnd: timetableItem.timeEnd)
                    }
                }
                self.tableView.reloadData()
            } else {
                self.alert.showAlertAccordingToStatusCode(fromController: self, statusCode: status!)
            }
        })
    }

    private func getUserGroupe() {
        let keychainManager = KeychainManager()
```

```
        let key = keychainManager.readUserSecretAPIkeyFromKeychain().data(using:
String.Encoding.utf8)!
        let userSecretAPIkeyBase64Encoded = key.base64EncodedString(options: [])
        let header = ["Authorization" : "Basic \(userSecretAPIkeyBase64Encoded)"]
        userService.getUser(headers: header, completionHandler: { userItem, status in
            guard let status = status else {
                log.warning("Status code is nil.")
                return
            }

            if status == 200 {
                self.groupe = userItem?.groupe
            } else {
                self.alert.showAlertAccordingToStatusCode(fromController: self, statusCode: status)
            }
        })
    }
}
```

Б.8 Вихідний код функцій для керування файловим сховищем даних

```
import Alamofire
class FileService {
    let decoder = JSONDecoder()

    func getAllFolders(completionHandler: @escaping ([File]?, Int?) -> Void) {
        Alamofire.request(Constants.BASE_URL + "/folders").responseDecodableObject(keyPath:
nil, decoder: decoder) {
            (response: DataResponse<[File]>) in
            let status = response.response?.statusCode
            switch response.result {
            case .success:
                guard let status = status else {
                    log.error("Request passed without status code - status code is nil.")
                    completionHandler(nil, nil)
                    return
                }

                guard status == 200,
                    let folderData = response.result.value else {
                        log.debug("Request passed with status code, but not 200 OK: \(status)")
                        completionHandler(nil, status)
                        return
                }
                completionHandler(folderData, status)

            case .failure(let error):
                guard let status = status else {
                    log.error("Request failure with error: \(error.localizedDescription)")
                    completionHandler(nil, nil)
```

```
                return
            }
            log.error("Request failure with status code: \(status) and error:
\(error.localizedDescription)")
            completionHandler(nil, status)
        }
    }
}
func getAllFilesByFolderTitle(folderTitle: String, completionHandler: @escaping ([File]?,
Int?) -> Void) {
    Alamofire.request(Constants.BASE_URL +
"/folders/\(folderTitle)/files").responseDecodableObject(keyPath: nil, decoder: decoder) {
        (response: DataResponse<[File]>) in
        let status = response.response?.statusCode
        switch response.result {
        case .success:
            guard let status = status else {
                log.error("Request passed without status code - status code is nil.")
                completionHandler(nil, nil)
                return
            }

            guard status == 200,
                let fileData = response.result.value else {
                    log.debug("Request passed with status code, but not 200 OK: \(status)")
                    completionHandler(nil, status)
                    return
            }
            completionHandler(fileData, status)

        case .failure(let error):
            guard let status = status else {
                log.error("Request failure with error: \(error.localizedDescription)")
                completionHandler(nil, nil)
                return
            }
            log.error("Request failure with status code: \(status) and error:
\(error.localizedDescription)")
            completionHandler(nil, status)
        }
    }
}

func createNewFile(file: File, folderTitle: String, completionHandler: @escaping (File?, Int?) -
> Void) {
    let fileToJSON = Mapper().toJSON(file)
    Alamofire.request(Constants.BASE_URL + "/folders/\(folderTitle)/files",
                      method: .post,
                      parameters: fileToJSON,
                      encoding: JSONEncoding.default).responseObject {(response:
DataResponse<File>) in
                if let status = response.response?.statusCode {
```

```
                switch(status) {
                case 200:
                    guard response.result.isSuccess else {
                        completionHandler(nil, response.result.value?.error);
                        log.error("Error: REST request to create file has failed. ", context:
response.result.value?.error)
                        return
                    }
                    completionHandler(response.result.value?.value, nil)
                    log.debug("REST request to create file has passed successfully.")
                default:
                    completionHandler(nil, response.result.value?.error);
                    log.error("Error: REST request to create file has failed. ", context:
response.result.value?.error)
                }
            }
        }
    }

    func updateFile(file: File, completionHandler: @escaping (File?, Int?) -> Void) {
        let fileToJSON = Mapper().toJSON(file)
        Alamofire.request(Constants.BASE_URL + "/folders/\(folderTitle)/files",
                          method: .put,
                          parameters: fileToJSON,
                          encoding: JSONEncoding.default)
            .responseObject {(response: DataResponse<File>) in
                if let status = response.response?.statusCode {
                    switch(status) {
                    case 200:
                        guard response.result.isSuccess else {
                            completionHandler(nil, response.result.value?.error)
                            log.error("Error: REST request to update category has failed. ", context:
response.result.value?.error)
                            return
                        }
                        completionHandler(response.result.value?.value, nil)
                        log.debug("REST request to update category has passed successfully.")
                    default:
                        completionHandler(nil, response.result.value?.error)
                        log.error("Error: REST request to update category has failed. ", context:
response.result.value?.error)
                    }
                }
        }
    }
}
```